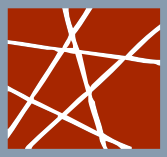

Network Architectures
and Services
NET 2013-07-1

Dissertation

Analysis and Control of Middleboxes in the Internet

Andreas Müller

Network Architectures and Services
Department of Computer Science
Technische Universität München

TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Analysis and Control of Middleboxes in the Internet

Andreas Müller

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Hans-Arno Jacobsen
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Georg Carle
	2.	Prof. Dr. Ernst W. Biersack (Institut Eurécom, Sophia Antipolis, Frankreich)

Die Dissertation wurde am 18.12.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 26.04.2013 angenommen.

Cataloging-in-Publication Data
Andreas Müller
Analysis and Control of Middleboxes in the Internet
Dissertation, July 2013
Network Architectures and Services, Department of Computer Science
Technische Universität München

ISBN 3-937201-35-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
DOI 10.2313/NET-2013-07-1
Network Architectures and Services NET 2013-07-01
Series Editor: Georg Carle, Technische Universität München, Germany

ABSTRACT

With the growing size and complexity of the Internet several types of middleboxes have been introduced to the network in order to solve a number of urgent problems. Network Address Translation devices fight against the Internet address depletion problem, caches and proxies help to efficiently distribute content and firewalls protect networks from potential attackers. Unfortunately, middleboxes violate the end-to-end connectivity model of the Internet protocol suite and therefore introduce numerous problems for services and applications. The contribution of this thesis is the study and development of concepts and algorithms for understanding and improving the behavior of middleboxes, their traversal, as well as their application to problems that emerged with the growing success of the Internet with a focus on unmanaged networks such as home and small company networks.

The first part of this thesis designs and implements tools and algorithms to analyze middlebox behavior. We introduce a processing model for describing functional characteristics and create an information model to structure relevant middlebox behavior parameters. As an experimental analysis, a public field test is conducted to evaluate existing traversal techniques, understand middlebox behavior, to gain knowledge about their deployment and to draw conclusions on how to improve middlebox traversal.

The second part of this thesis focuses on the traversal of middleboxes. Existing middlebox traversal methods do not differentiate between different types of applications and therefore deliver suboptimal results in many situations. We claim that the classification of applications into service provisioning categories helps to determine the best matching middlebox traversal technique, not only dependent on the network topology, but also considering user-defined requirements. This thesis presents a framework that improves the communication of existing and future applications and services across middlebox devices. The idea is to use previously acquired knowledge about middlebox behavior and services for setting up new connections. Obtained results show that the knowledge-based approach is not only more flexible and applicable, but due to the decoupled connectivity checks, also significantly faster compared to the state of the art. Based on the findings of our experimental analysis we show how to integrate and adapt existing middlebox traversal techniques into the middlebox traversal framework. Additionally, new middlebox traversal techniques are presented and evaluated.

In the third part of this thesis the applicability of middlebox services to unmanaged networks is discussed. A secure service infrastructure is presented based upon a trust model combining the power of centralized Certificate Authorities with the flexibility of the Web of Trust and social networks. This security infrastructure fulfills the security requirements of the middlebox traversal framework and provides secure identities for users, services and hosts in a semi-automatic way. A middlebox service helps establish and coordinate secure communications between different domains according to user-defined access control policies. Three application examples for middleboxes show how existing services can be improved and extended to solve open security, privacy and connectivity issues.

KURZFASSUNG

Mit der zunehmenden Komplexität heutiger Netzwerke werden Funktionalitäten zur Lösung aktueller Probleme häufig als Middlebox im Netzwerk implementiert. Network Address Translation (NAT) hilft mit der limitierten Anzahl von IPv4 Adressen umzugehen, Firewalls filtern ungewollten Verkehr und Proxies und Web-Caches sind für die effiziente Verteilung von Inhalten unerlässlich. Da Middleboxen jedoch das initiale Ende-zu-Ende Prinzip des Internets verletzen, führt deren Existenz zu einer Vielzahl von Problemen mit existierenden Protokollen, Anwendungen und Diensten. In dieser Arbeit wird das Verhalten von Middleboxen sowie deren Konsequenz für heutige Netzwerke untersucht. Darauf aufbauend werden Methoden und Algorithmen entwickelt, die eine verhaltensbasierte Traversierung von Middleboxen ermöglichen. Darüber hinaus werden aktuelle Sicherheits- und Privatheitsprobleme aufgezeigt und Middlebox-basierte Lösungen dafür vorgestellt.

Im ersten Teil der Arbeit werden Methoden und Algorithmen entworfen, die in der Lage sind, verschiedene Verhaltensweisen von Middleboxen zu analysieren. Anhand zweier Modelle, einem Verarbeitungs- und einem Informationsmodell, wird das Verhalten in einem Feldtest anschließend experimentell untersucht. Die so gewonnenen Ergebnisse helfen Schlussfolgerungen bezüglich der Traversierung von Middleboxen zu ziehen, existierende Techniken dafür zu verbessern sowie neue Techniken zu entwerfen.

Im zweiten Teil der Arbeit wird ein wissensbasiertes Framework entwickelt, das eine anforderungsgerechte, sichere und performante Middlebox-Traversierung für verschiedene Anwendungsklassen ermöglicht. Existierende Traversierungsmethoden betrachten einige explizite Anforderung von Applikationen nicht, beispielsweise bezüglich der Sicherheit, was zu ungewollten Seiteneffekten führen kann. In dieser Arbeit werden Anwendungsklassen für Applikationen entworfen um Traversierungstechniken nicht nur basierend auf der Netzwerktopologie, sondern auch basierend auf der eigentlichen Anwendungskategorie auszuwählen. Kernpunkt des entwickelten Frameworks ist ein wissensbasierter Ansatz, der das Verhalten von Middleboxen sowie zusätzliche Regeln analysiert, geeignete Traversierungstechnik parameterisiert und damit nicht nur zuverlässiger, sondern auch performanter als der Stand der Technik ist.

Im dritten Teil der Arbeit werden neue Lösungskonzepte für die Anwendbarkeit von Middlebox-basierten Diensten auf aktuelle Sicherheits- und Privatheitsprobleme vorgestellt. Insbesondere in Netzwerken, die nicht professionell administriert sind, wie beispielsweise Heimnetzwerke, können aktuelle Sicherheitsmechanismen aus Komplexitätsgründen oft nicht eingesetzt werden. Diese Arbeit entwickelt eine Sicherheitsinfrastuktur, die herkömmliche Zertifizierungsstellen (CAs) mit dem Ansatz des Web of Trust verbindet und für unerfahrene Benutzer zugänglich macht. Abschließend wird anhand von drei neuen Middlebox-Diensten exemplarisch gezeigt, wie sich aktuelle Sicherheits- und Privatheitsprobleme mit Middlebox-basierten Diensten lösen lassen.

ACKNOWLEDGMENTS

This thesis would have not been possible without the support of many individuals:

First of all I would like to thank Prof. Dr.-Ing. Georg Carle for giving me the opportunity to work on many interesting projects during the course of my studies and for supervising this thesis. Thank you Prof. Dr. Ernst Biersack for being my second supervisor and for the valuable feedback and Prof. Dr. Hans-Arno Jacobsen for the organization of the examination.

Main parts of this work are part of the research project AutHoNe - Autonomic Home Networking, funded by the Federal Ministry of Education and Research (BMBF). Their financial support was an important foundation.

I would like to thank my colleagues at the chair for Network Architectures and Services for creating a productive work environment and for many discussions on various subjects. In particular I would like to thank my former colleague Dr. Andreas Klenk for many discussions on NAT and Holger Kinkelin for his cooperation on the security part and for sharing our office with me for the past 5 years. I also had the chance to work with numerous good students and student assistants who contributed to this work.

Thank you Ralph Holz and Dr. Richard Edel for reading and correcting this thesis and for giving valuable feedback on the English language and on the topic itself. A big thanks to my friends who provided me a world outside of research, computer science and work.

I am also very thankful to my parents who have always supported and believed in me. Without their support throughout my life many opportunities would not have been possible. Finally, I would like to thank my wife Rebecca, not only for constantly reading and correcting this thesis, but especially for being there when I needed her (and for her good cooking and baking). Without her, this thesis would not exist.

München, July 2013

Andreas Müller

Previously published parts of this thesis:

- A. Müller, G. Münz, and G. Carle. Collecting Router Information for Error Diagnosis and Troubleshooting in Home Networks. In the workshop WISE, held in conjunction with the IEEE Conference on Local Computer Networks 2011 (LCN), Bonn, Germany, October 2011.
- A. Müller, H. Kinkelin, S.K. Ghai, and G. Carle. A Secure Service Infrastructure for Interconnecting Future Home Networks based on DPWS and XACML. In HomeNets: ACM SIGCOMM Workshop on Home Networks, New Delhi, India, September 2010.
- A. Müller, N. Evans, C. Grothoff, and S. Kamkar. Autonomous NAT Traversal. In 10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2010), Delft, The Netherlands, August 2010.
- A. Müller, H. Kinkelin, S.K. Ghai, and G. Carle. An Assisted Device Registration and Service Access System for future Home Networks. In IFIP Wireless Days 2009, Paris, France, December 2009.
- A. Müller, A. Klenk, and G. Carle. ANTS - A Framework for knowledge-based NAT Traversal. In IEEE Globecom 2009 Next-Generation Networking and Internet Symposium, Honolulu, Hawaii, USA, November 2009.
- A. Müller, A. Klenk, and G. Carle. Behavior and Classification of NAT devices and implications for NAT Traversal. IEEE Network Special Issue on Implications and Control of Middleboxes in the Internet, October 2008.
- A. Müller, A. Klenk, and G. Carle. On the Applicability of knowledge-based NAT Traversal for future Home Networks. In Proceedings of IFIP Networking 2008, Singapore, May 2008.

CONTENTS

Part I Introduction and Background 1

1. Introduction . . . 3
1.1 Thesis Objectives and Research Questions . . . 4
1.1.1 Positioning and Goals . . . 4
1.2 Contributions and Document Structure . . . 6

2. Background . . . 9
2.1 End-to-end Principle of the Internet . . . 9
2.2 Analysis of today's Internet . . . 11
2.3 Middleboxes . . . 12
2.3.1 Introduction . . . 12
2.3.2 Categories and Properties . . . 13
2.3.3 Impact and Problems . . . 13
2.3.4 Assessment . . . 14
2.4 Operation of selected Middleboxes . . . 15
2.4.1 Network Address Translation . . . 15
2.4.2 Large Scale NAT . . . 16
2.4.3 Firewalls and ALGs . . . 19

Part II Analysis and Behavior of Middleboxes 21

3. Modeling and Classification of Middlebox Behavior . . . 23
3.1 Introduction . . . 23
3.2 Modeling of Middlebox Behavior . . . 23
3.2.1 Related Work . . . 23
3.2.2 Notation and Processing Model . . . 24
3.2.3 Modeling Network Address Translation . . . 25
3.3 Classification of NAT Behavior . . . 26
3.3.1 NAT Behavior for Outgoing Packets . . . 27
3.3.2 Incoming Packets . . . 30
3.3.3 NAT Classification . . . 30
3.3.4 Behavior of Large Scale NATs . . . 32
3.4 Firewall Behavior . . . 33
3.4.1 Stateless Firewalls . . . 33
3.4.2 Stateful Firewalls . . . 34
3.5 Application Layer Middleboxes and Proxies . . . 34
3.5.1 Tor and Polipo . . . 34
3.6 Summary and Key Findings . . . 35

4. Experimental Analysis of Middlebox Behavior . . . 37
4.1 Introduction . . . 37
4.2 Information Model . . . 38
4.2.1 Stateful Element . . . 38
4.2.2 Filtering Element . . . 39
4.2.3 Translation Element . . . 40
4.2.4 Reference Example of a Middlebox Instance . . . 40
4.3 Detailed Description of our Measurement Algorithms . . . 41
4.3.1 Behavior Measurements . . . 43
4.3.2 Behavior-based Traversal Analysis . . . 49
4.3.3 Additional Measurements . . . 50
4.3.4 Topology Measurements . . . 52
4.3.5 Active Monitoring of Middlebox Parameters . . . 55
4.4 Experiments in the Lab and Verification of our Algorithms . . . 56
4.4.1 Test Setup . . . 56
4.4.2 Virtualized Testbed and Topology Generator . . . 56
4.5 Field Test . . . 58
4.5.1 Related Work . . . 59
4.5.2 Requirements and Contributions . . . 59
4.5.3 Design and Implementation . . . 60
4.6 Results and Discussion . . . 64
4.6.1 Testset Description . . . 64
4.6.2 Binding and Filtering Behavior Results . . . 67
4.6.3 Mapping Behavior Results . . . 72
4.6.4 Additional Behavior Results . . . 77
4.6.5 Behavior-based Traversal Results . . . 80
4.6.6 Additional Results . . . 81
4.6.7 Topology Results . . . 82
4.6.8 Lessons Learned . . . 83
4.7 Summary and Key Findings . . . 85

Part III Traversal of Middleboxes 87

5. Middlebox Traversal and Service Provisioning . . . 89
5.1 Introduction . . . 89
5.1.1 Middlebox Traversal Problem . . . 90
5.2 State of the Art in Middlebox Traversal . . . 92
5.2.1 Explicit Solutions . . . 92
5.2.2 Behavior-based Solutions . . . 94
5.2.3 Additional Solutions . . . 98
5.2.4 Evaluation and Comparison . . . 99
5.3 Application-Centric Middlebox Traversal . . . 101
5.3.1 Service Categories for Middlebox Traversal . . . 101
5.3.2 Application of Existing Traversal Techniques . . . 102
5.4 Summary and Key Findings . . . 104

6. Knowledge-based Middlebox Traversal . . . 105
6.1 Introduction . . . 105
6.2 Knowledge-based Middlebox Traversal . . . 106
6.3 Reference Examples for a knowledge-based Framework . . . 107
6.3.1 Unilateral Deployment . . . 107
6.3.2 Coordination of Traversal through Signaling . . . 107
6.4 NOMADS: A new Middlebox Traversal Framework . . . 110
6.4.1 Scenarios . . . 110
6.4.2 Architectural Overview . . . 112
6.4.3 Signaling Module: Request Response Protocol . . . 113
6.4.4 Application Interfaces . . . 115
6.4.5 Integration and Adaption of Middlebox Traversal Techniques . . . 116
6.4.6 Decision Module . . . 118
6.4.7 Implementation . . . 123
6.5 New Middlebox Traversal Techniques . . . 123
6.5.1 Devices Profiles for Web Services IGD . . . 123
6.5.2 Autonomous Middlebox Traversal . . . 127
6.6 Evaluation and Discussion . . . 130
6.6.1 Adaption to Experimental Results . . . 131
6.6.2 Applicability Evaluation . . . 132
6.6.3 Performance Evaluation . . . 133
6.7 Summary and Key Findings . . . 136

Part IV Security and Application of Middleboxes 137

7. A Secure Service Infrastructure for Unmanaged Networks . . . 139
7.1 Introduction . . . 139
7.2 Survey of the State of the Art . . . 140
7.2.1 Identity Management for Unmanaged Networks . . . 140
7.2.2 Trusted Third Party . . . 140
7.2.3 Web of Trust . . . 141
7.3 Architectural Overview and Components . . . 141
7.3.1 Requirements and Contributions . . . 142
7.3.2 Components . . . 143
7.4 Identity Management . . . 144
7.4.1 Identities . . . 144
7.4.2 Device Registration . . . 144
7.5 Trust Establishment . . . 146
7.5.1 Direct Pairing . . . 146
7.5.2 Remote Pairing using Social Networks . . . 146
7.6 Authorization . . . 148
7.6.1 SSL/TLS interception for legacy Services . . . 149
7.7 Implementation . . . 150
7.7.1 Hardware-based Security Extension . . . 151
7.7.2 Integration . . . 151
7.8 Possible Attacks and Recommendations . . . 152
7.9 Summary and Key Findings . . . 153

8. New Middlebox Services . . . 155
8.1 Introduction . . . 155
8.2 A Middlebox for securing DPWS . . . 155
8.2.1 Technology Overview . . . 156
8.2.2 Scenarios and Contributions . . . 157
8.2.3 Approach . . . 157
8.2.4 DPWS service usage across Networks . . . 158
8.2.5 Security Discussion . . . 161
8.2.6 Evaluation . . . 162
8.2.7 Summary . . . 163
8.3 PrivMid6: A Privacy Preserving Middlebox for IPv6 . . . 164
8.3.1 Related Work . . . 165
8.3.2 Scenarios . . . 166
8.3.3 Requirements . . . 166
8.3.4 Design of PrivMid6 . . . 167
8.3.5 Evaluation and Discussion . . . 169
8.3.6 Summary . . . 171
8.4 Virtual and Dynamic Infrastructures . . . 172
8.4.1 Application Areas . . . 172
8.4.2 Approach . . . 173
8.4.3 Virtualized Networks - P2PVPN . . . 174
8.4.4 Resource Manager . . . 181
8.4.5 Summary . . . 183
8.5 Summary and Key Findings . . . 184

Part V Conclusions 185

9. Conclusion . . . 187
9.1 Contributions . . . 187
9.2 Future Work . . . 191

Appendix 193

A. DTD of the Middlebox Information Model . . . 195

B. List of Middleboxes . . . 199
B.1 List of Recommended Home Routers . . . 199
B.2 List of Non-Recommended Home Routers . . . 201

C. NOMADS Documentation . . . 203
C.1 NOMADS Decision Tree . . . 203
C.2 NOMADS Signaling Protocol . . . 204
C.3 NOMADS Socket API . . . 204

Bibliography . . . 207

Part I

INTRODUCTION AND BACKGROUND

1. INTRODUCTION

Within the last 20 years the Internet has developed from a small controllable computer network for a limited group of people to a mass medium with more than 2.4 billion participants worldwide [75]. This network is not only used for sending emails, reading websites and sharing pictures, but also as a basis for many commercial applications with a predicted worldwide revenue of almost $1trl. in 2013 [73]. This trend continues with the availability of affordable mobile devices and data plans, social networks and business models such as online music (iTunes), electronic books (Amazon) and on-demand video streaming (Netflix). It is not hard to predict that online activities will continue to grow with the availability of smart energy grids and smart meters, interconnected cars and home appliances.

As the Internet was being designed and evolved, one of the most important design decisions was the end-to-end principle. It states that application specific functions "can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system" [136]. The intermediate network itself is seen as an unreliable transportation medium holding as little state as possible. When new functionality becomes available it should be integrated into the end-systems and distributed among the participating hosts.

With the growing number of participants and the growing heterogeneity of hardware and software, it has become almost impossible to establish new functionality on a large scale by relying on the end-hosts to update to a new version of the operating system, the application or the communication stack. Therefore, devices providing additional functionality to the network and its users have been introduced and placed as intermediate nodes in the "middle" of communication paths: middleboxes. The most famous examples for such devices are Network Address Translation (NAT) boxes, firewalls and transparent proxies. Most of them were introduced for solving a specific problem, e.g. the address depletion problem in case of NAT, and although they violated the initial end-to-end paradigm, they have been accepted due to missing alternatives.

The violation of the end-to-end paradigm through middleboxes introduces numerous problems to existing networks. Since middleboxes are generally transparent to end-hosts, many applications don't work the way they're supposed to work. Hosts located behind stateful middleboxes are not reachable from the outside and security policies for firewalls are too static for complex application protocols using in-band signaling and realm specific transport addresses [24]. Many solutions to such problems have been developed [1, 48, 128, 130], but because of the lack of standardized middlebox behavior many of them only deliver suboptimal results and do not consider application and user-specific requirements.

Future scenarios such as home, car and building networks call for new functionalities and again cause a number of challenges when connecting them to existing networks. Many of these challenges, such as privacy, security, quality of service and rich content distribution, can be solved by well-defined and well-understood middleboxes. The definition and acceptance of such new middlebox services can be seen as an important step for middleboxes to become a valid design principle for future networks.

1.1 Thesis Objectives and Research Questions

Middleboxes provide more and more functionality to today's networks, while at the same time creating a number of problems caused by the violation of the end-to-end paradigm. The objective of this thesis is to develop concepts and algorithms to discover, analyze and cope with middleboxes in the Internet, to understand their impact and to minimize their negative consequences. Our main claim is that only a solid understanding of the network, the topology, the existence and the behavior of middleboxes allows to design algorithms and solutions for coping with the negative side-effects they introduce. Thus, this thesis provides answers to the following questions:

Q1 Does a thorough and structured analysis of middlebox behavior help to understand the implications for applications and services and to improve the success rate of middlebox traversal techniques? (corresponding Chapters 3 and 4)

Q2 Can the knowledge about the behavior of involved middleboxes be used to apply and parameterize middlebox techniques suitable for applications and the involved infrastructure? (Chapters 5 and 6)

Q3 Can security mechanisms common in enterprise networks be applied to networks that lack professional administration in order to secure middlebox traversal? (Chapter 7)

Q4 Can well-designed and well-understood middlebox services be a valid design principle for the Internet and can middlebox services be applied to open problems in today's networks? (Chapter 8)

1.1.1 Positioning and Goals

Middlebox research areas can be divided into two main categories: The first one investigates how legacy protocols can work across existing middleboxes. Part of this problem is the **Discovery and Monitoring of Middleboxes** to understand their existence and impact before actually designing solutions for the **Configuration, Control and Traversal of Middleboxes**. The second category examines how new protocols and middleboxes can be designed in order to overcome the problems of today's deployment. The **Security and Application of Middleboxes** is therefore an important step towards an end-middle-end design principle for future Internet architectures.

Discovery and Monitoring of Middleboxes

Current research activities on middlebox discovery can be structured into two categories: The first one treats middleboxes as black boxes and assumes no direct control. For example, STUN [130] allows discovering the public endpoint of a connection by querying a STUN server, thus revealing parts of the mapping behavior of possible NATs on the path. The second category defines new interfaces for middleboxes and specifies protocols for controlling them. NSIS [64] and the "TCP Option for Transparent Middlebox Discovery" [84] are two examples for this category.

The goal of **Part II**, **Analysis and Behavior of Middleboxes**, is to understand middlebox behavior by an experimental analysis. Our approach considers and analyzes relevant behavioral parameters as identified from related work and from our own observations. Based on these parameters, an information model is designed and algorithms and concepts are developed to reveal specific behavior patterns. After an evaluation in our lab, the algorithms are used to conduct a public field test to shed light on the actual situation in today's networks and to give recommendations on how to improve

middlebox traversal. Part of this analysis is the design and implementation of an algorithm for discovering cascaded middleboxes, as well as our Lightweight Information Export tool LinEx, a data collection agent for actively monitoring middlebox behavior.

Configuration, Control and Traversal of Middleboxes

Once discovered, existing approaches aim to traverse middleboxes either directly (control-based traversal) or indirectly (behavior-based traversal). When assuming direct control, middleboxes are no longer an obstacle, but provide generic interfaces for configuring them based on requirements for the connection. NUTSS [61], an end-middle-end architecture, is an example for such an approach. Other research activities describe a "path coupled signaling for NAT/FWs" (NSIS NSLP) [147] by extending the NSIS protocol [64]. As an additional step towards controlling and configuring, [121] defines "Managed Objects for Middlebox Communication" in order to control middleboxes in a standardized way. Behavior-based traversal include approaches such as hole punching [48] and ICE [128].

The goal of **Part III, Traversal of Middleboxes** is to design solutions for middlebox traversal. We argue that if we consider the combination of middlebox behavior, the requirements for the application, as well as external policies, we will be able to select the best matching traversal technique for a given situation. In some cases, and if available, signaling via a third party can help to agree on a method and to negotiate relevant behavior parameters that can be used as input for the actual traversal technique. While our middlebox traversal framework is able to treat middleboxes on the path as black boxes and traverses them based on their behavior, it also provides interfaces for expressing access on a high-level. Furthermore, our DPWS [42] enabled Internet Gateway Device allows the automatic and secure discovery and configuration of middleboxes.

Security and Application of Middleboxes

To be compliant with authorization requirements of applications, security mechanisms for controlling middleboxes and frameworks are needed. State of the art security mechanisms, such as Public-Key Infrastructures (PKI), are complex and hard to maintain, especially for inexperienced users that can be found in unmanaged networks, e.g. home and small company networks. Middleboxes as a valid design principle for a network architecture has already been proposed in the state of the art. The explicit end-to-middle authentication and authorization mechanisms in [70] use the cryptographic namespace of the Host Identity Protocol (HIP) [98] in order to allow middleboxes to directly interact with end-hosts. [111] presents an end-to-middle security approach for the Session Initiation Protocol (SIP) [131]. With this approach a user agent is able to implement a mixture of end-to-end and hop-by-hop security by disclosing information only to those intermediate hosts that actually need it.

The goal of **Part IV**, **Security and Application of Middleboxes**, is to improve the security of unmanaged networks and to design new middlebox services for solving existing problems in today's networks. We present a secure service infrastructure that allows the management of secure identities in a user-friendly way and hides the complexity of existing protocols and operations from its users. Additionally, it presents new middlebox services for solving existing problems such as privacy concerns with IPv6, interconnecting home networks in an automatic way and solutions for managing virtual infrastructures.

1.2 Contributions and Document Structure

The contribution of this thesis is the study and development of concepts and algorithms for understanding the behavior of middleboxes, their traversal, as well as their application to problems that emerge with the growing success of the Internet. Table 1.1 shows the individual contributions and the methodology for their evaluation. It also links to the corresponding research questions of section 1.1 and numbers the contributions. This numbering is then used throughout the thesis and the individual contributions are described in more detail at the end of each chapter when presenting the chapter's key findings.

Chapter	Contribution	Software Description	Model	Emulation	Testbed Experiments	Field Test
Part II Analysis and Behavior of Middleboxes (Q1)						
3	(C3.1) Processing Model for Middleboxes		•			
4	(C4.1) Information Model for Middlebox Behavior		•			
	(C4.2) Algorithms for behavioral Measurements	•		•	•	
	(C4.3) Experimental Analysis of MB Behavior	•				•
Part III Traversal of Middleboxes (Q2)						
5	(C5.1) Service Categories for Middlebox Traversal		•			
6	(C6.1) Knowledge-based Framework for MB Traversal	•			•	•
	(C6.2) New Middlebox Traversal Techniques	•			•	•
Part IV Security and Application of Middleboxes (Q3+4)						
7	(C7.1) Security Infrastructure for Unmanaged Networks	•	•			
8	(C8.1)-(C8.3) Innovative Middlebox Services	•			•	

Tab. 1.1: Contributions of this thesis

This thesis is structured into five parts: I) Introduction and Background, II) Analysis and Behavior of Middleboxes, III) Traversal of Middleboxes, IV) Security and Application of Middleboxes and V) Conclusions. In the following we briefly explain the individual contents of the parts and their chapters:

Part I gives an introduction and explains the most relevant technologies for the understanding of this thesis. **Chapter 2** introduces the initial idea of the end-to-end paradigm of the Internet. We discuss today's Internet topology and present reasons why this paradigm is not completely true anymore. Middleboxes are introduced and typical categories and properties for classifying them are presented. We focus on the impact of middleboxes to existing protocols and applications and describe common problems they have introduced. We then present the operation of the most relevant middleboxes for this thesis, namely Network Address Translation (NAT), Large Scale NAT (LSN) and firewalls.

Part II analyzes middlebox behavior. The goal is to understand current middlebox behavior and deployment in order to use this knowledge for coping with them. In **Chapter 3** a model for describing the packet processing of specific middleboxes is presented (Contribution 3.1). Based on this model selected behavior issues according to the state of the art are described. **Chapter 4** then analyzes existing middlebox behavior in detail. An information model containing relevant parameters is designed (Contribution 4.1) and algorithms for testing these parameters are presented (Contribution 4.2). The results of an extensive field test using these algorithms give an insight on the deployment and behavior of middleboxes in today's Internet and allow us to give recommendations on how to improve existing traversal solutions and how to design future ones (Contribution 4.3).

Part III designs solutions for middlebox traversal. **Chapter 5** starts by describing the state of the art in middlebox traversal. We show why middlebox traversal should also take higher-level requirements, such as accessibility permissions, into account for selecting appropriate techniques based on these requirements. Thus, service categories for applications are defined and state of the art techniques are put in relation to them (Contribution 5.1). Based on these service categories, **Chapter 6** introduces our knowledge-based approach for middlebox traversal (Contribution 6.1). The framework NOMADS supports new and legacy applications that utilize our service categories, it is more reliable and due to the decoupled knowledge gathering it also introduces less latency than state of the art frameworks. We show how to integrate and improve existing traversal techniques by parametrizing them based on the actual behavior of all involved middleboxes. Finally, two new middlebox traversal techniques are presented: AutoMID (Autonomous Middlebox Traversal) allows hole punching [48] without the support of a third party and DPWS-IGD implements a secure Internet Gateway Device based on the Devices Profile of Web-Services (Contribution 6.2).

Part IV delivers concepts for the security and application of middleboxes. **Chapter 7** presents a security infrastructure for unmanaged networks (Contribution 7.1). The goal is to assist inexperienced users when introducing security mechanisms to their networks. Identity management, trust establishment, authorization and authentication solutions are discussed. **Chapter 8** then presents individual middlebox services targeting open problems of today's networks. First, a proxy solution in combination with an application layer firewall for securely interconnecting web-services is presented (Contribution 8.1). Secondly, an approach for enabling privacy in IPv6 networks based on middleboxes is developed (Contribution 8.2). The last section presents virtualized infrastructures and proposes a combination of host and network virtualization to deploy and maintain flexible and secure network configurations (Contribution 8.3).

Chapter 9 of **Part V** concludes this thesis by summarizing its contributions and gives an outlook for future work.

2. BACKGROUND

When the Internet appeared as a communication network back in the 1980's, the growth and expansion of the network we have today was hardly imaginable. However, with the principle design of the Internet the architects of the network had already calculated that it would only scale if the functionality of the network was reduced to the minimum to overcome performance and cost issues. The so-called end-to-end argument states that the network itself should only implement functions that support the basic transmission of packets from A to B. Additional functionality, such as reliability, error recovery, security and loss-detection should only be implemented at the end of the communication path. The end-to-end principle is deemed to be the driving factor for today's popularity of the Internet since its basic design supports a manifold number of heterogeneous applications, devices and users. However, the pure end-to-end argument is not true anymore. Due to different restrictions and many competing players, it may sometimes only be possible to implement new functionality in the middle of the network. Security threats, economical decisions and deployment considerations are just a few reasons. The introduction of middleboxes not only provides additional functionality, but due to the violation of the end-to-end argument middleboxes also cause many problems that have been discussed in the research community over the last few years.

This chapter gives an introduction to fundamental terminology and describes technologies that are important for the understanding of this thesis. It also helps put this work into the context of the state of the art and emphasizes the importance of research in this area. We first describe the idea of the initial end-to-end principle in section 2.1 and give background on how the end-to-end principle became one of the main reasons for the success of today's Internet. Afterwards, we analyze today's Internet topology and describe why the pure end-to-end argument is not true anymore. Section 2.3 then introduces the concept and classification of middleboxes in general. We explain the basic operation of common middleboxes in section 2.4 that are most important for this thesis: Network Address Translation devices (NAT) and their successor Large Scale NAT (LSN) for solving the IPv4 address depletion problem, as well as firewalls for enforcing security policies.

2.1 End-to-end Principle of the Internet

Different to real networks, such as the postal network, computer networks send digital copies of data chunks from A to B. If a chunk gets lost on its way, it can easily be replaced and sent again. In networks that are dealing with real goods, e.g. packages or letters, each intermediate node has to take special care of the part to be delivered since it cannot be replaced. This fact was essential for the design principles of the Internet.

The authors of [136] called their principle the "end-to-end argument" and they suggested that a function implemented at a lower-level of the system may either be redundant (because it will be implemented on a higher-level as well) or extremely costly for the value it provides. Additionally, they stated that certain functionality can only be implemented correctly "with the knowledge and help of the application standing at

the end points of the communication system." As an example, a file transfer between two hosts can only be reliable if both hosts actively check the integrity of the files and notify the other end if an error is detected. If a reliability mechanism would be implemented on a lower-level in the network as well, it would be redundant and may cause performance issues. Another example is the implementation of security mechanisms. Although intermediate nodes may be able to check the authenticity of messages, the application of the receiver will still have to recheck it again. Thus, it would not only introduce an enormous overhead by performing the same task multiple times, it would also generate compatibility and upgrade problems: If the applications decide to switch to a new version of the integrity check algorithm, this would only be possible if all intermediate hosts also participate. The end-to-end principle became the fundamental basis of the Internet, which can be seen as "a packet switched communications facility in which a number of distinguishable networks are connected together using packet communications processors called gateways which implement a store and forward packet forwarding algorithm" [26]. The Internet Engineering Task Force (IETF) also released an informational document about the architecture of the Internet [15] where they list a number of general design issues to be considered. Most of them comply with the end-to-end principle and can only be fulfilled with it. Heterogeneity of devices, underlying networks and software call for a modular approach. Trade-offs between performance, cost and functionality were already cornerstones of the authors motivation for postulating the end-to-end principle in [136]. [91] is not only looking at the technical aspect of the end-to-end principle, the author states that the success of today's Internet is related to the end-to-end principle. The design decision allows the easy introduction of new applications, services and users without being able to decide which ones are good or bad. The rather "dumb" network cannot differentiate "between a packet carrying Republican speech and a packet carrying Democratic speech", thus it is unable to discriminate certain users and eventually supports innovations through network neutrality.

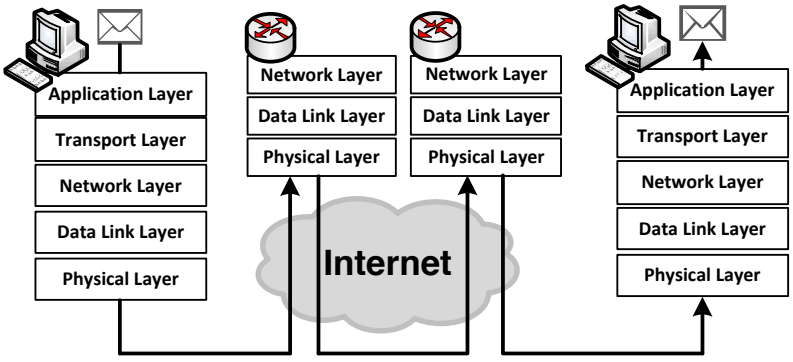

Fig. 2.1: The end-to-end design principle

Figure 2.1 depicts the end-to-end paradigm and shows how packets travel between two hosts. The application on the left sends the message which finds its way through the different layers. The only functionality of intermediate hosts, or routers, is to forward the packet to the next hop according to their routing table. Intermediate nodes therefore operate on the network layer and only the final destination passes the packet up to the application layer. In this example the protocol on the transport layer, e.g. TCP, would be responsible for a reliable data transmission. Thus, reliability is clearly implemented in the end-hosts of the communication.

2.2 Analysis of today's Internet

When taking a closer look at the end-to-end principle it becomes clear that the definition is somehow vague and a connection can never be purely end-to-end. The file transfer example as described above aims to send data from A to B in a reliable way. But what happens between the user and the part where data chunks get fragmented and sent to the network? Maybe there were problems when writing data to the disk or the operating system lost packets due to other problems. What is actually the definition of "end"? And what about email communication? Following the pure end-to-end principle an email would be sent directly from host A to B, while implementing all email specific functionality only on hosts A and B. In reality, email servers are used as intermediate relays or proxies that take care of storing and distributing mails on behalf of their users. However, this "violation" is only theoretical.

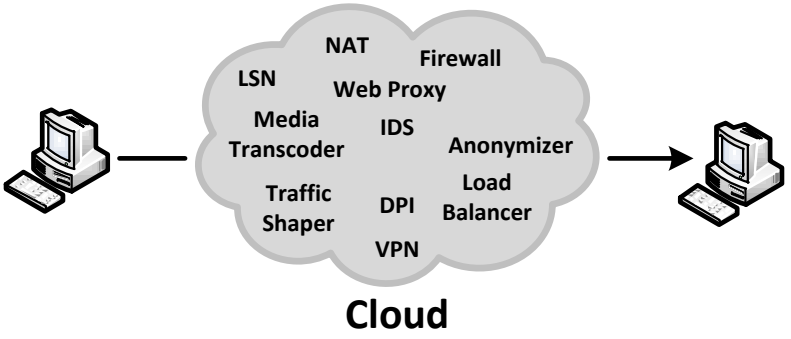

Fig. 2.2: Deployment of middleboxes in today's Internet

Figure 2.2 shows the current situation of today's Internet. Instead of only having intermediate nodes that take care of layer 3 routing, a large number of devices have appeared that provide additional functionality on multiple layers. Routers may only forward packets after performing a Deep Packet Inspection (DPI), firewall policies regulate traffic flows according to their security policies and Load Balancers help to distribute available resources equally.

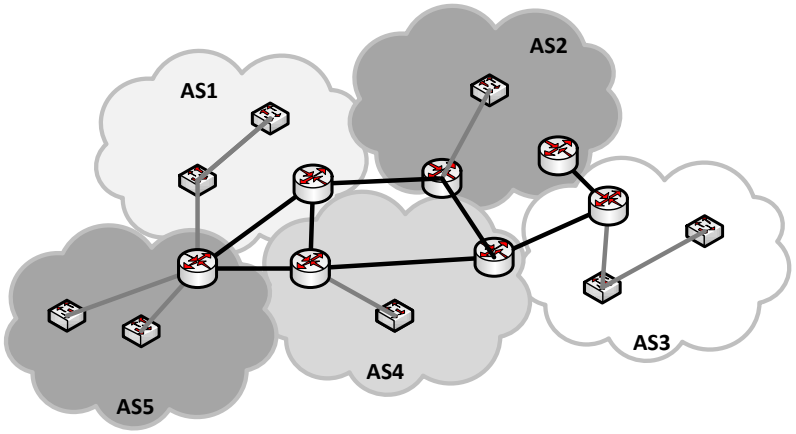

Fig. 2.3: Example topology found in today's Internet

But what are the reasons for the existence of these devices that clearly violate the initial end-to-end argument that is seen as one of the most important success

factors of today's Internet? One reason is that the Internet has developed from a flatly organized academic network into a clustered network as depicted in figure 2.3. Instead of being one network, today's Internet is a compilation of many autonomous networks (AS), where each is under a different independent administrative control. Commercial Internet Service Providers (ISP) run their networks as a business and expect a profit from it [82]. The interconnection of these systems is therefore not driven by performance or goodwill, but rather by peering contracts and mutual (legal) agreements. Each administrator aims to protect his network and cares about all packets that travel through it. Furthermore, some administrators even aim to block unwanted traffic and assign different levels of quality of service to packets that do not comply with their policy, leading to many debates and discussions about network neutrality [29, 65, 156].

A second reason for introducing intermediate devices was the lack of security. In the early days of the Internet security was completely neglected since only trusted hosts participated in the rather small network. With the introduction of e-commerce, online banking and the interconnection of company networks, security became an essential requirement, often solved by introducing additional functionality, such as firewalls, Intrusion Detection Systems (IDS) and Deep Packet Inspection (DPI), into the network.

Another initial argument for the end-to-end argument was the protection of innovation. In the early days of the Internet, changes in the network were considered costly and updating end-hosts was still reasonable. However, with the large number of users we see today, this argument has turned around: Changes on the end-host systems are considered almost impossible since many of them cannot or will not be updated by their users. Introducing functionality to the network seems to be the only way of not raising compatibility issues and reaching a quick deployment of new functionality.

Altogether, the clustering of the Internet, the independency of autonomous systems and the change from a small academic network to a multi-billion dollar business were primary success factors of today's Internet. However, the introduction of additional functionality to the network and the violation of the initial end-to-end principle also led to suboptimal routes, additional delays and to a undefined behavior of many protocols. For example, [66] studied 142 access networks in 24 different countries and measured the impact of intermediate devices to TCP connections. Their tests revealed that many of these connections were modified in an unordinary way, which is a strong indicator for the existence of black boxes located in the middle of the network.

2.3 Middleboxes

The last sections mentioned some important reasons for establishing additional functionality in the middle of a communication path rather than on the end-hosts. This section gives an introduction to middleboxes. We first present an overview of middlebox functionality and try to extract common properties in order to categorize them. Afterwards, we discuss problems that the rise of middleboxes introduce and analyze the impact on existing applications and protocols. Finally, we assess the presented facts.

2.3.1 Introduction

Middleboxes are defined as "intermediary devices performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host" [17]. Middleboxes operate on different layers, from the network layer to the application layer, and may drop, insert, transform or modify

packets traveling through it. Some implementations, such as data relays or web proxies, are also capable of terminating IP packet flows and originating new ones according to their policy. Thus, middleboxes are never the end of a communication session, but were introduced to provide additional functionality to the network, such as caching, quality of service, filtering and address translation.

2.3.2 Categories and Properties

RFC 3234, Middleboxes: Taxonomy and Issues [17], gives an overview about current middleboxes in the Internet, their functionality and their impact on protocols and applications. The document defines eight facets of middleboxes related to their functionality, deployment, processing and behavior. These properties are then used to describe and categorize a large number of middleboxes that are known to exist. The following listing presents a selection of these facets and table 2.1 gives an overview of existing middleboxes. We added four main reasons for their deployment and identified the most important and most common middleboxes today.

1. The **Layer of Operation** describes on which layer of the ISO/OSI model the middlebox operates. Some of them might only work up to the network layer (simple firewall not considering ports), while others also consider the transport layer (Network Address and Port Translation (NAPT)) or even the application layer (Application Layer Gateway (ALG)).

2. **Transparency** of the middlebox: is the middlebox part of the protocol **(N**ot-**T**ransparent) or is it completely transparent for it **(T**ransparent).

3. What is the purpose of the middlebox: **Functional (F) vs. Optimizing (O)**. Functional means the middlebox is part of the application, which cannot work without it, whereas optimizing means that the middlebox is just an addition to the protocol trying to optimize it.

4. Does the middlebox only forward packets (**Routing (R)**) or does it actually change them (**Processing (P)**)?

5. Does the middlebox work in a **Stateful (SF) or Stateless (SL)** way? Stateful means that the middlebox has to maintain an entry for packet flows and will forward related packets according to this entry. If no entry exists, the packet cannot be processed (e.g. incoming packet for NAT). The functionality of the middlebox is then limited by the number of entries it can handle. With a stateless implementation the middlebox is always able to decide about the processing based on the packet itself.

2.3.3 Impact and Problems

Although middleboxes provide new functionality to networks, services and users, the violation of the end-to-end argument has a huge impact on many protocols. One of the first documents that described these problems was RFC 2775 [16]. Released in 2000, the authors are worried about the transparency of the Internet and state that it is “impossible to estimate with any numerical reliability how widely the above inventions have been deployed.” The term “above inventions” refers to middleboxes and the problem they saw was the in-transparency they are causing: “Since many of them preserve the

Reason for Introduction	Example Middlebox	Properties according to listing above Layer	Transp.	Purp.	Oper.	State
IP Address Depletion	NAT44 [140]	3+4	T	F	P	SF
	NAT44 with ALG	3-7	T	F	P	SF
Address Independency	NPTv6 [159]	3	T	F	P	SL
Security	Firewall	3+4	T	F	R	SF
	Firewall (DPI)	3-7	T	F	R	SF
	Tunnel Endpoint	3-4	T	F	P	SF
	SOCKS [90]	4-7	NT	F	R	SF
	Anonymizer	3-7	T	F	P	SL
Reliability	Load Balancer	3-7	T	F	R	SL or SF
Performance	Caching	7	T or NT	F+O	P	SL
	Proxy	7	T or NT	F+O	P	SL
Innovation	Media Transcoder	7	T or NT	F+O	P	SL or SF

Tab. 2.1: Existing middleboxes, the reason for introducing them as well as their properties

illusion of transparency while actually interfering with it, they are extremely difficult to measure."

[17] describes and analyzes the "current impact of middleboxes on the architecture of the Internet and its applications." The problem mentioned is that every new middlebox that becomes available challenges older protocols that are not aware of it. One example is the introduction of peer-to-peer networks that are popular for sharing files in a client-to-client (peer-to-peer) manner. Today, the most prominent example using the peer-to-peer paradigm is Voice over IP (VoIP). While signalization and authentication uses a centralized server (e.g. a SIP proxy), the actual connection between the caller and the callee is peer-to-peer. Especially stateful middleboxes, such as NATs and firewalls, cause the protocol to fail because incoming packets towards a client will most likely be blocked due to a missing state. State is usually only created for outgoing packets in order to forward the reply to the initial initiator of the communication. Generally speaking, the connection establishment will more and more be initiated from within stub domains (the ISPs customers), making the existing problem even more severe.

Finally, although middleboxes help to deploy new functionality in many cases, they may also block innovations in others. For example, the introduction of new transport layer protocols, such as SCTP [144] and DCCP [85], is hard because existing middleboxes (e.g. consumer NATs) simply cannot translate the protocol as they don't implement it. In other cases poor implementations of protocol standards in middleboxes cause an unpredictable behavior in the protocol flow, which may lead to network stack crashes or other unrecoverable errors.

2.3.4 Assessment

There are three main reasons for the high number of middleboxes that are deployed in today's networks: First, the advantages of deploying new functionality in the network

without changing the end-hosts are stronger than the negative impact of middleboxes for current protocols and applications. Additionally, some functionality, e.g. stateful packet filtering, is only possible to implement at network borders instead of implementing it end-to-end. Secondly, at the time when a specific middlebox was deployed (e.g. NAT) no other solution solving the urgent problem that was the driving factor for the deployment (e.g. address depletion) was available. Once an alternative became available (e.g. IPv6), it was already too late. This will most likely happen again and again since innovations aiming for a quick deployment are hard to implement at end-hosts only. Finally, one last reason was mentioned by Mark Handley: "Whatever you think your problem is, plenty of box vendors will sell you a solution" [66]. Thus, there is no way of not accepting middleboxes as an elementary part of today's Internet.

2.4 Operation of selected Middleboxes

This section describes the basic operation of typical middleboxes, Network Address Translation devices, firewalls and Application Layer Gateways that are widely deployed in today's network.

2.4.1 Network Address Translation

In 1991, RFC 1287 [27] was the first to mention the limited address space of IPv4: "The Internet will run out of the 32-bit IP address space altogether, as the space is currently subdivided and managed". Although the introduction of Classless Inter-Domain Routing (CIDR) [52, 53] enabled a more efficient allocation of IP addresses, this was only a short term solution since the number of devices was still continuously growing. [27] also proposed to "replace the 32 bit field with a field of the same size but with different meaning. Instead of being globally unique, it would now be unique only within some smaller region [...]. Gateways on the boundary would rewrite the address as the packet crossed the boundary." [157] first mentioned dividing the address space into internal and external addresses, before [123] finally defined the private address space as we know it today. [44] describes how to translate IP addresses at the border of so-called stub domains (networks that do not handle transit traffic). The advantage of this solution is that internal addresses can be reused in other private domains, thus solving the address depletion problem. This technique became known as Network Address Translation (NAT). More background on the historical introduction of NAT and the reasons for the IETF for not standardizing it properly are well described in [167].

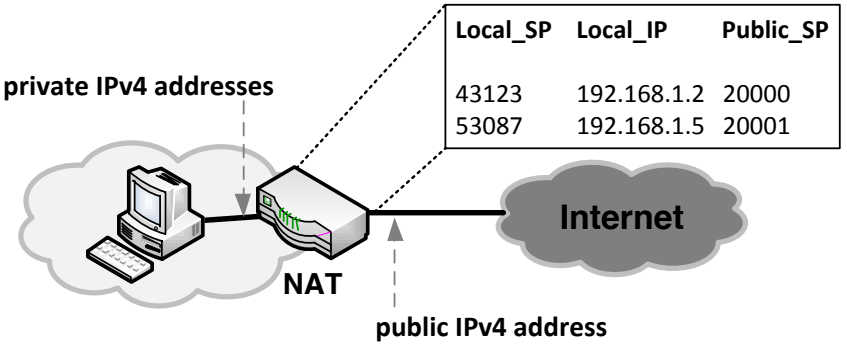

Fig. 2.4: Network address and port translation

Today, we distinguish between two types of NAT. With basic NAT for IPv4 (NAT44) as described in [44], only addresses are translated between two domains. The same is true for the IPv6-to-IPv6 Network Prefix Translation [159], which in comparison to NAT44 uses a checksum neutral mapping. An advanced version of NAT, Network Address and Port Translation (NAPT) [140], uses port numbers as an additional multiplexer, which allows the assignment of only one public address to a private network, while still allowing them to share it among a large number of devices. Today, most home networks use a NAPT to connect their devices to the Internet. Therefore, the more general term NAT will used in this thesis to cover both varieties, NATs and NAPTs.

In order to translate between a private and a public domain, the NAT device has to maintain a state for all connections. For every new connection attempt (e.g. a TCP SYN packet) coming from an internal host, the NAT creates a new entry in its mapping table. In NAT terminology this entry is called a mapping or binding [142]. As depicted in figure 2.4, the NAT stores the internal IP address and source port, as well as the external source port in its table. Dependent on the implementation, the NAT may also store the destination IP address and port to differentiate if incoming packets actually belong to an existing connection. The NAT then replaces the local source port by the external source port and the private source IP address by the public IP address of the NAT. Additionally, the checksum of the layer 3 and layer 4 headers (for TCP and UDP) have to be recalculated. For incoming packets the NAT looks up its mapping table using the destination source port as a key and retranslates the destination port and IP address to the private one.

NAT not only solves the address depletion problem, it also has two additional advantages: First, NAT helps to hide the topology of the internal network, which can be seen as a security plus. Second, the address independency of the internal network allows changing the ISP without changing the network configuration of all hosts and vice versa. On the other hand, NAT breaks the end-to-end argument of the Internet and due to the stateful behavior, it causes many problems, which are further described in section 5.1.1.

Part of the problem is that although published as early as 1994 [44], no common approach for NAT has emerged. Current NAT implementations differ from model to model, which leads to many compatibility issues. It is therefore essential to understand the behavior of NAT in order to design solutions that cope with the disadvantages of it. The details of NAT behavior are covered in part II.

2.4.2 Large Scale NAT

Although almost every consumer network (and also many enterprise networks) uses NAT to connect to the Internet, the Internet Assigned Numbers Authority (IANA) has run out of global IPv4 addresses. The transition to IPv6 is hard because a large number of hosts in private networks still only support IPv4 and many consumer electronic devices, such as Internet radios, smart TVs and smartphones, will never receive a firmware update supporting IPv6. Therefore, providing IPv6-only services to new customers is not possible.

The second problem is that most content in the Internet is not yet reachable via IPv6. In fact, according to the Comcast IPv6 adaption monitor[1] only 3.06% of the Alexa[2] top one million websites was reachable via IPv6 from their monitoring clients in May 2012. A second test in December 2012 revealed that according to Comcast's

[1] http://v6monitor.kangaroo.comcast.net:8180/monitor/
[2] http://www.alexa.com

AAAA and IPv6 connectivity statistics[3] 4.948% of the top 1M websites provide an AAAA record. Although the number of IPv6 enabled websites is growing, the challenges for ISPs for the future can be summarized as follows:

- ISPs run out of IPv4 addresses and cannot provide public IPv4 addresses to all of their customers.
- ISPs have to provide access to the IPv4 Internet to all of their customers (from IPv4 and IPv6 hosts).
- ISPs have to support legacy IPv4-only devices in the customers network.
- ISPs have to be able to provide access to the IPv6 Internet, as well as to support IPv6 devices in the future.

Large Scale NAT (LSN) (sometimes also Carrier Grade NAT (CGN)) is an approach to shift NAT functionality from the customer to the ISP. The assumption is that many customers don't need more than a few thousand simultaneous connections, which means they will not need a global IPv4 address and use up the full range of 2^{16} ports for multiplexing. Some architectures tend to deploy address translation only at the provider, while others are based on cascaded and multi-layered NATs. Altogether, LSN can be seen as an interesting approach for solving the IPv4 depletion problem and may delay the full transition to IPv6 for many more years. The following sections give an overview of different protocols and possibilities how to deploy LSNs. The IETF RFC 6264 [78] gives an incremental approach for combining these protocols to an incremental LSN. Chapters 3 and 6 will then discuss and analyze the behavior of LSNs and the problems they introduce (section 3.3.4) and show algorithms that allow discovering (section 4.3.4) and traversing (section 6.4.5) them.

NAT444 and NAT464

The first and more or less obvious approach for saving IPv4 addresses is to introduce multiple layers of NAT to the network. NAT444 means that instead of assigning public IPv4 addresses to the customers NAT router (ISPs refer to the term Customer Premises Equipment (CPE)), customers only receive a private IP address from the ISP. The translation into a public address is then done by a NAT device located at the provider. This scenario is shown in figure 2.5. The provider may have a large block of public IPv4 addresses (e.g. from the IANA-reserved range [160]), which is assigned to the public interfaces of one or more LSNs.

NAT444 is the easiest way to support new customers. It's immediately available and there are no changes at the CPE required. Instead of translating between a private and a public address, the CPEs translate between two private addresses using the same algorithm as before. Besides general problems with LSN (e.g. some applications may fail, see section 3.3.4), NAT444 has some major drawbacks: First, overlapping addresses have to be avoided. This means customers have to make sure that they don't use the same private address space on the LAN and the WAN side of their NAT. Furthermore, some CPEs filter private addresses on the WAN side, which means they cannot be used in a NAT444 scenario. The firewall functionality of such devices simply block all incoming packets carrying a private source address. To avoid these problems, it would be possible to declare a range of public IP address as "ISP shared" and reuse these addresses within the providers networks. However, this requires a lot of administrative effort, which makes this solution unlikely to happen.

[3] http://www.employees.org/~dwing/aaaa-stats/

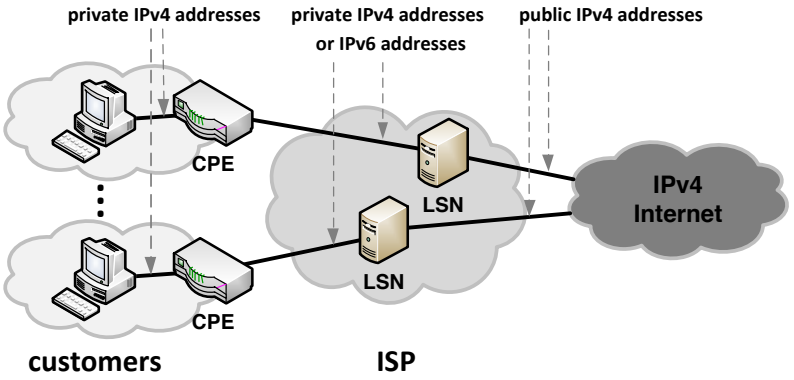

Fig. 2.5: NAT444 and NAT464

As depicted in figure 2.5, NAT464 uses public IPv6 addresses between the CPE and the LSN. The problem here is that the CPEs have to implement an address translation between IPv4 and IPv6 addresses. Thus, the ISPs would have to replace all their deployed CPEs. The initial approach for translating between IPv4 and IPv6 was standardized as NAT-PT in [153] and because of "technical and operational difficulties" obsoleted in [3]. The successor of NAT-PT, the Framework for IPv4/IPv6 Translation, is standardized in [7].

Dual-Stack Lite (DS-Lite)

The initial idea for the transition from IPv4 to IPv6 was to assign two addresses to each host: an IPv4 address and an IPv6 address operating as a Dual Stack. However, since ISPs are running out of IPv4 addresses, Dual-Stack Lite (DS-Lite) only assigns IPv6 addresses to customers. IPv6 devices may then use these addresses to directly connect to the Internet. In order to also support IPv4 hosts, the CPE is also able to assign private IPv4 addresses to its connected hosts. To be more precise, the private IP addresses are coming from the ISP, while the CPE only chooses the subnet mask for the private network. On outgoing packets, the CPE encapsulates the IPv4 packet into an IPv6 packet and uses its IPv6 connection to send it to the ISP. The Address Family Transition Router element (AFTR) [43] of the ISP decapsulates the packet, translates the private IPv4 address into a public one and sends the packet to the public IPv4 Internet. DS-Lite was mainly pushed by Comcast and is now standardized in [43]. The advantage of DS-Lite is that there is no need for translating network layer protocols and it allows deploying IPv6 in the ISP network while still supporting IPv4 connectivity and IPv4 customers. As IPv6 devices become available they can be directly connected without the need for a tunnel.

NAT64 together with DNS64

Finally, ISPs have to provide access from IPv6 only devices to content that is only accessible via IPv4. Whenever a hosts asks a DNS64 server [6] for an AAAA record that does not exists, the DNS server checks if an A record (IPv4) exists for the domain. If so, it constructs an AAAA record in a way that the first part of the returned IPv6 address points to a NAT64 device, while the second part embeds the IPv4 address of the A record. The initial packet is then sent to the LSN, which extracts the IPv4 address and translates the packet to an IPv4 source address. [8] defines the well-known prefix

64:ff9b::/96 to be used for embedding addresses. Stateful NAT64 is standardized in [5] and [7], DNS64 in [6]. Viagenie[4] published an open source implementation called Ecdysis[5] integrating NAT64 and DNS64.

2.4.3 Firewalls and ALGs

In the early days of the Internet there were only a few users and security critical attacks or fake messages were nothing to worry about. As the number of users and hosts continued to grow security became a critical issue. Anonymous users from all over the world, online businesses, financial transactions and e-commerce are just a few examples for motivating network security. The easiest way for blocking unwanted traffic is to install a filter on the edge of the network and require all packets to go through it. Once a policy for a specific packet exists, it is either dropped or forwarded. Firewalls normally work on layer 3 and 4, thus considering IP addresses and port numbers. For some protocols Application Layer Firewalls or Application Layer Gateways (ALG) also exist. Some of them also look into the payload of certain packets, which is referred to as Deep Packet Inspection (DPI).

Stateless Packet Filters treat each packet individually, there is no relation between them. For every packet, incoming and outgoing, that aims to traverse the firewall, the administrator has to define an explicit policy. Protocols using in-band signaling with dynamic connections (e.g. FTP, SIP) will certainly fail since the firewall cannot know that these packets actually belong to a valid session.

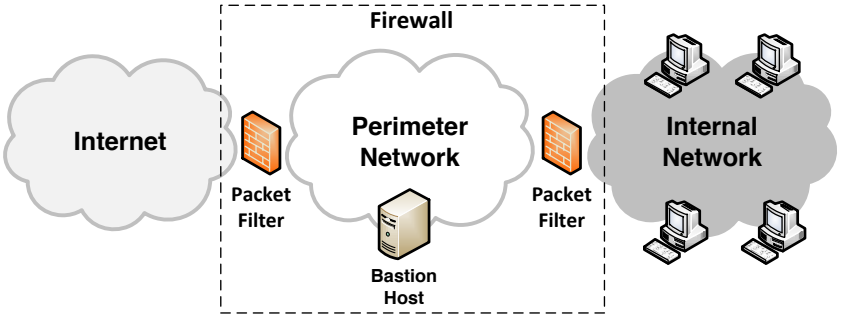

Fig. 2.6: Screened subnet firewall architecture

Today, most firewalls are implemented as **Stateful Packet Filters** and keep track of the state of the protocol (e.g. the TCP state). Instead of adding rules for both directions (host A is allowed to communicate with host B on port 80 and host B is allowed to send packets from port 80 to host A) a stateful firewall only knows one rule: host A is allowed to communicate with host B using http. Packets that are part of the answer are detected as related to an established connection and are automatically forwarded. Thus, host B can only send packets to host A if they are actually part of an answer and only if host A actually established a connection. Thus, spoofing becomes hard as the attacker has to guess the correct parameters of a packet (e.g. sequence number) in order to traverse the firewall. For stateless protocols such as UDP, most firewalls add their rules dynamically and a small timeout (e.g. 30-60 sec.) is assigned to each connection. Once the timeout expires, the state is deleted. For TCP the timeout

[4] http://www.viagenie.ca
[5] http://ecdysis.viagenie.ca

is much larger (10-30 min.), since a termination of a session can be easily detected by tracking the TCP state (e.g. waiting for a FIN packet).

For protocols that use dynamic ports, for example SIP [131], SDP [67], RTSP [138] and FTP [120], a firewall only operating on layer 3 and 4 would have to be configured to allow a large port range, thus creating a potential security risk. Instead, an Application Layer Gateway is able to look into the layer 4 payload and handle the application layer protocol. The firewall is then able to dynamically open the correct port for this application. An ALG for NAT is also able to replace IP addresses and port with a local scope by the corresponding public ones, thus helping the protocol to seamlessly work across NATs.

Figure 2.6 shows a typical firewall deployment in an enterprise network following the screened subnet architecture. This architecture differentiates between two types of hosts: regular clients and bastion hosts. Bastion hosts provide services to the network and need to be reachable from the Internet. One example would be an SMTP server that is responsible for the company's mail infrastructure. In order to separate bastion hosts and regular clients the screened subnet architecture consists of two networks: An internal network for regular clients that do not provide any services to other hosts and a perimeter network for bastion hosts. Packet filtering routers are located on the edge of each network and only forward packets that are explicitly allowed by the company's policy. If the bastion host is compromised it is still isolated from the internal network and an attacker is not able to sniff other packets or directly attack internal hosts. However, if an attacker manages to compromise an internal client, a perimeter firewall only helps prevent establishing connections towards the Internet. In this case a personal firewall installed on each host may be an additional security benefit.

Part II

ANALYSIS AND BEHAVIOR OF MIDDLEBOXES

3. MODELING AND CLASSIFICATION OF MIDDLEBOX BEHAVIOR

3.1 Introduction

Most protocols that exist today were designed with the end-to-end paradigm in mind. Middleboxes in the Internet may violate this paradigm and therefore cause errors in the original and intended packet flow. Firewalls that block certain ports and understand only a limited number of protocols, load balancers that choose a path for packets in an unforeseen manner and Network Address Translators that rewrite IP addresses and port numbers make it hard to deploy new protocols and applications that still assume direct connectivity from one end-host to another, e.g. Voice over IP and peer-to-peer applications such as Bittorrent[1].

In order to assess the impact of existing middleboxes, it is essential to understand and describe their mode of operation. Thus, the goal of this chapter is to design a processing model that allows describing the processing of packets for arbitrary middleboxes. Based on this model as described in section 3.2, section 3.3 gives a detailed summary of middlebox behavior with a focus on Network Address Translators and firewalls, as these devices have the most impact on all layers, from the network layer up to the application layer.

3.2 Modeling of Middlebox Behavior

A middlebox operates in the "middle" of a communication session by modifying, filtering or adding packets that are seen on its network interfaces according to middlebox specific processing rules. The approach is quite generic: For example, a traditional firewall has to decide whether a packet received on interface A should be forwarded to interface B or if it should be discarded. A Network Address Translator receives packets on its internal LAN-side interface, translates them and forwards them using its external WAN-side interface. A Load Balancer receives a query and decides to which instance of a server it should forward the requests. However, when closely looking at the middlebox specific functionality, it is rather hard to precisely describe the processing functionality of the individual implementations. This section introduces a notation for describing the packet processing in middleboxes that is useful to describe, analyze and classify middlebox behavior.

3.2.1 Related Work

Due to the complexity of today's Internet topology, there has been much research on modeling and formalizing network communications, architectures and also middleboxes in particular. In [81] the authors present a model "to express precisely and abstractly the concepts of naming and addressing and to specify a consistent set of control patterns and operational primitives, from which a variety of communication services can

[1] http://www.bittorrent.org/

be composed". Their model allows constructing general network architectures and also includes basic middlebox functionality and operations. In fact, the existence of middleboxes and their undefined and multi-layered way of operation was part of the motivation for this work. As a runtime environment the authors used the Click Modular Router [86], which itself is a prominent example for modeling traffic flows.

As already described in section 2, RFC 3234 [17] gives a detailed overview of middleboxes that have emerged in different networks and describes their functionality. Other related work on specific middlebox behavior is done by the BEHAVE working group of the IETF[2]. There are a number of drafts [134] and RFCs [4, 59, 141] that group and characterize different aspects and requirements of middlebox behavior and give an overview of important fields to be considered.

A specific model describing a Unified Firewall Model is presented in [108]. The authors list and combine firewall specific attributes and their model allows describing the syntax and semantics of typical, but also complex firewall implementations.

An approach of modeling middleboxes in general was presented in [80]. With the help of zones, input pre-conditions, processing rules, state databases, auxiliary traffic and interest fields the model is able to describe the different phases of a middlebox when processing packets. Middlebox specific functionality, such as the assignment strategy for a new NAT mapping, is not considered. Additionally, the complexity of the model specification is rather high. The model we use to describe the state of the art in middlebox behavior has a strong focus on middlebox specific functionality and aims to be less complex that the state of the art.

3.2.2 Notation and Processing Model

The processing model is based on the assumption that a middlebox forwards packets from one domain (or zone) to another. More specifically, a middlebox receives packets on zone A, processes them and sends them to zone B. As in Linux *iptables*, we use three different processing stages to express this (see figure 3.1): Input (In), Processing (Proc.) and Output (Out). The input stage receives packets and if the packet is destined for the middlebox it triggers functions in the processing stage where middlebox specific functionality is implemented. For example, once a firewall receives a packet it queries its policy table and decides whether the packet should be accepted and forwarded or if the packet should be discarded. After processing the packet it is sent to the output stage (if allowed by the policy) and forwarded to the external zone.

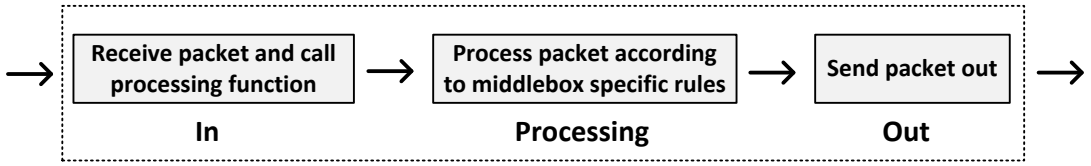

Fig. 3.1: Processing stages of a middlebox

Each middlebox implements a generic function at the input stage that forwards all received packets to the appropriate processing function. Middlebox specific processing rules are then implemented in the processing stage, which may (dependent on the policy) call the *send* function in the output stage. Our notation consists of variables (e.g. p) that may be passed to functions (e.g. $function(p)$). If a function manipulates

[2] http://www.ietf.org

variable p we state: $p := function(parameter)$. The following table lists the packet flow through the individual stages and the main functions for our model. In the following section we then focus on the middlebox specific functionality as implemented in the processing stage.

In	**if** $packet\_arrived(p)$ **then**
	$Proc(interface, p)$
	end if

Proc	implements MB specific rules in function $Proc(interface, p)$ which may call a function in the output path
Out	$send(p, interface)$

3.2.3 Modeling Network Address Translation

Network Address Translation devices are probably the most common middleboxes today and are installed in almost every consumer network. The following example shows how a basic Network Address Translation (NAT) device can be modeled. For the sake of clarity, we use two different listings, one for outgoing (algorithm 3.1) and one for incoming packets (algorithm 3.2). Outgoing packets arrive on the internal interface (thus $Proc(internal, p)$ has to be implemented). The NAT looks up its state table to determine if the packet belongs to an existing connection. If so, it retrieves the appropriate entries for this connection from the state database (the external port and IP address) and translates the relevant header fields (e.g. source IP address, source port and checksums). If no entry exists, the NAT has to allocate a new external mapping, store it to the database and modify the packet accordingly. The following listing shows the individual steps of a NAT for **outgoing packets** using our processing model.

```
1 if found (pub := get_state(p)) then
2 |   p := translate_packet(p, pub);          ▷ translate packet immediately
3 else
4 |   pub := allocate_new_mapping(p);         ▷ a new external mapping is needed
5 |   store_state(p);                          ▷ remember new mapping
6 |   p := translate_packet(p, pub);           ▷ translate packet
7 end if
8 send(p, external);                          ▷ call send function in output stage
```

Alg. 3.1: $Proc(internal, p)$ for NAT outgoing packets

The individual processing steps for the function $Proc(internal, p)$ are as follows:

1. The NAT mapping table is queried (*get_state*) with the packet p as a key to check if the packet belongs to an existing connection. It expects the external public mapping *pub* in return (line 1).
2. If the connection is already known to the NAT mapping table the packet is translated according to NAT specific rules implemented in the *translate_packet* function (line 2).

3. If the connection is not known to the NAT (line 3) a new external public mapping (*pub*) is allocated following a strategy that is described later in this chapter. Here we simplify this step by using the *allocate_new_mapping* function that expects the packet p as a parameter and returns the public mapping *pub* (line 4).
4. The NAT stores the new state to its state table and translates the packet (as before) (lines 5+6).
5. Finally, the NAT calls the *send* function in the outgoing stage and passes packet p and the external interface *external* to it (line 8).

For **incoming packets** (p arrives at the *external* interface, thus $Proc(external, p)$ has to be implemented) the NAT is only able to forward packets if they belong to an existing connection. Algorithm 3.2 shows the processing steps for incoming packets.

```
1 if found (priv := get_state(p)) then
2 |   p := translate_packet(p, priv);          ▷ translate packet immediately
3 |   send(p, internal);                       ▷ call send function in output stage
4 else
5 |   drop_packet(p);                                          ▷ drop packet
6 end if
```

Alg. 3.2: $Proc(external, p)$ for NAT incoming packets

The individual processing steps for the function $Proc(external, p)$ are as follows:

1. The NAT mapping table is queried (*get_state*) with the packet p as a key to check if the packet belongs to an existing connection. It expects the internal mapping with the private addresses *priv* in return (line 1).
2. If the connection is known to the NAT mapping table the packet is translated according to NAT specific rules implemented in the *translate_packet* function (line 2).
3. After the translation the NAT calls the *send* function in the outgoing stage and passes packet p and the internal interface *internal* to it (line 3).
4. If the connection is not known to the NAT (line 4) it drops the packet (line 5).

3.3 Classification of NAT Behavior

Although standardized at the very beginning [44], current NAT implementations not only differ from vendor to vendor, but also from model to model and sometimes even from firmware to firmware. As a result, an application might work across a particular NAT, while it fails with a different model or version. Therefore, it is essential to understand and classify existing NAT implementations to design algorithms and applications that have a high chance of working with a large number of existing NATs. In the last few years, a lot of research on classifying NAT behavior has been done. The following sections present the state of the art in NAT behavior following the terms and definitions of [133] and [4]. Additional protocol specific behavior requirements have been described in [59] (for TCP), [4] (for UDP) and [141] (for ICMP). Table 3.1 gives an overview about relevant behavior categories and their properties. The meaning of the properties are then described in the next sections.

Category	Sub Category	NAT Property
Binding	Port Binding	Port Preservation Port Multiplexing No Port Preservation
	NAT Binding	Endpoint Independent Address (Port) Dependent Connection Dependent
Filtering		Independent Address Restricted Address and Port Restricted

Tab. 3.1: Overview of relevant NAT behavior categories

3.3.1 NAT Behavior for Outgoing Packets

For outgoing packets that do not belong to an existing connection (meaning no state exists in the mapping table), the NAT has to allocate a new mapping, store it to the state table and finally translate the mapping according to the NATs policy. There are a number of options for allocating a new state, which then have an impact on the storing and translating parameters.

Allocate Mapping

As shown in listing 3.1 at line number 4 the NAT calls the *allocate_new_mapping* function for every packet that it has not yet seen. The main goal of the allocation process is to create a mapping that uniquely identifies the current connection. Possible multiplexers are the source port of the external zone, the IP address of the external zone (if multiple exist), or a combination of fields such as the source and the destination port together with the layer 4 protocol. The allocation function can be divided into *port binding* and *NAT binding*.

A port binding behavior of *Port Preservation* means that the NAT will try to keep the source port and only translate the source IP address. The following listing shows how port preservation can be expressed using our general middlebox notation. The relevant changes to the example of section 3.2.3 are highlighted.

```
1 if found (pub := get_state(p)) then
2 |   p.sIP := pub.IP;                          ▷ translate IP address only
3 else
4 |   pub := allocate_new_mapping(p);
5 |   store_state(p);
6 |   p.sIP := pub.IP;                          ▷ translate IP address only
7 end if
8 send(p, external);
```

Alg. 3.3: $Proc(internal, p)$ for NAT using Port Preservation

If simple port preservation is not possible, because the mapping would not be unique anymore, the extension *Port Multiplexing* may be used in order to still preserve the port. Here, multiplexing is achieved by also considering the destination address of the packet. Incoming packets can now carry the same destination port and are distinguished by the complete 5 tuple. In this case the NAT is able to assign the same external source port and external source IP address multiple times. The NAT may also follow the strategy of not preserving any source ports. Here, we distinguish between a *No Port Preservation* strategy that uses a defined *algorithm* for assigning external ports and a strategy that simply uses more or less *random* external ports. While in the first case the predictability of external ports only depends on the algorithm, in the latter case port prediction is almost impossible.

NAT Binding deals with the dependency and reuse of mappings. *Endpoint Independent Binding* means that the assigned external mapping is only dependent on the source of the connection (port and IP address). As long as a host establishes a connection from the same source IP address and port, the mapping does not change independent of the destination. With an *Address (Port) Dependent Binding* the assignment is dependent on the internal and the external transport address (combination of IP address and port according to [130]). As long as consecutive connections from the same source to the same destination are established, the mapping does not change. As soon as a different destination is involved, the NAT changes the external multiplexing. An *Endpoint Dependent Binding*, or connection dependent binding, assigns a new port to every connection. Some implementations increase the new port number by a specific (and well predictable) delta, but others assign random port numbers to new mappings.

```
1  case port preservation
2  |  pub.sP := p.sP;                                  ▷ keep source port
3  case no port preservation (alg)
4  |  pub.sP := lastPort + X;                          ▷ use algorithm for new port
5  case no port preservation (random)
6  |  pub.sP := random(portrange);                     ▷ random port
7  case endpoint independent
8  |  pub.sP := alg(p.sP, p.sIP);                      ▷ use source as input
9  case connection dependent
10 |  pub.sP := alg(p.dP, p.dIP);                      ▷ use destination as input
11 end case
```

Alg. 3.4: Implementation of *allocate_new_mapping*(p) dependent on the NAT type

The listing above shows the implementation of the allocation function for different behaviors. One example of a no port preservation algorithm is to increase the last allocated port by a fixed value X. A random strategy would call a randomize function that picks a new port from the list of available ports. An endpoint independent binding uses an algorithm that only considers the packets source port and IP address, whereas a connection dependent mapping considers the destination of the packet.

Store and Update Mappings

The number of fields that have to be stored in the internal mapping table depends on the port allocation algorithm and on the filtering behavior (see below). For the simplest form of NAT it is sufficient to store the source IP address and port of the internal client, as well as the external port of the NAT. To manage existing mappings the NAT also assigns a timer to every connection. The timer is reset for every packet traveling through the NAT. If it expires the mapping is removed. For many protocols there are multiple timers. For example, UDP manages a single UDP packet timer, as well as a "UDP stream timeout" that is allocated after multiple packets belonging to the same connection (a stream) travel through the NAT. For TCP, every TCP-state (e.g. SYN-sent, established, ...) maintains its own timer, while it is recommended to set the value of the established state to approximately two hours [59].

Translating Packets: Masquerading

The main operation of a NAT device is to translate realm-specific addresses between two networks. The *translate_packet* function specifies which fields are translated. For a pure address translation device, such as traditional NAT [44] or NPTv6 (NAT Prefix Translation) [159], only the network layer is affected. The NAT replaces the source IP address and adjusts the checksums. In case of NPTv6, the translation is done in a checksum neutral way. In case of NAT the checksum of the layer 4 packet has to be recalculated as well since the IP source and destination addresses are part of TCPs pseudo header [119]. For a NAPT device that is not preserving ports the source IP address, source port and checksum have to be replaced. For NATs operating up to layer 7 for specific protocols, the *translate_packet* function also replaces the payload of the application layer protocol e.g. for SIP. Finally, some NATs also decrement the time to live (TTL) field of the IP header.

```
1  case Network Address Translation Device
2  |   p.sIP := pub.IP;                          ▷ replace source IP address
3  |   p := adjust_checksums(p);                 ▷ Layer 3 and 4 checksums
4  case Network Prefix Translation
5  |   p.sIP := pub.IP;                          ▷ Checksum neutral translation
6  case NAPT with port preservation
7  |   p.sIP := pub.IP;
8  |   p := adjust_checksums(p);
9  case NAPT no port preservation
10 |   p.sIP := pub.IP;
11 |   p.sP := pub.Port;                         ▷ replace source port
12 |   p := adjust_checksums(p);
13 case NAPT with SIP ALG
14 |   p.sIP := pub.IP;
15 |   p.sP := pub.Port;
16 |   p := adjust_checksums(p);
17 |   p := translate_SIP(p);                    ▷ Application Layer Gateway
18 end case
```

Alg. 3.5: Implementation of $translate\_packet(p, pub)$ dependent on the NAT type

3.3.2 Incoming Packets

Once a mapping exists, the NAT has to decide if an incoming packet actually belongs to an existing connection or if it only matches parts of the state by accident. *Endpoint Filtering* therefore describes how existing mappings can be used by external hosts and how NAT handles incoming connection attempts. *Independent Filtering* allows inbound connections without considering the source of the packet. As long as it matches an existing state the packet is forwarded to the appropriate internal client. With *Address Restricted Filtering* the NAT only forwards packets coming from the same host (matching IP address) the initial packet was sent to. *Address and Port Restricted Filtering* also compares the source port of the inbound packet. According to our middlebox notation, the three categories can be described as follows:

```
1  case independent filtering
2  |  if p.dP == state.pubPort then
3  |  |  return state.private_mapping;
4  |  end if
5  case address restricted filtering
6  |  if p.dP == state.pubPort AND p.sIP == state.dIP then
7  |  |  return state.private_mapping;
8  |  end if
9  case address and port restricted filtering
10 |  if p.dP == state.pubPort AND p.sIP == state.dIP AND p.sP ==
   |  state.dP then
11 |  |  return state.private_mapping;
12 |  end if
13 end case
```

Alg. 3.6: Implementation of *get_state*(p) dependent on the NAT type

3.3.3 NAT Classification

With the basic NAT behavior aspects described above, we will now discuss a categorization used in the state of the art and depicted in figure 3.2.

Full Cone NAT

A Full Cone NAT uses an endpoint independent binding strategy, which means that the same external endpoint (*ext* according to figure 3.2) is assigned to two consecutive connections from the same source transport address (*int*) independent from the destination transport address (*dest*1 and *dest*3). Additionally, a Full Cone NAT implements an independent filtering strategy, thus any external host is allowed to send packets from arbitrary ports (*destX*) to the established mapping. Since the NAT only needs to maintain three main entries (plus a few more such as a timer) for each connection, Full Cone NATs are inexpensive to build and widely used.

Address Restricted Cone NAT

Like a Full Cone NAT, an Address Restricted Cone NAT also uses an endpoint independent binding. However, address restricted filtering is used as the filtering strategy.

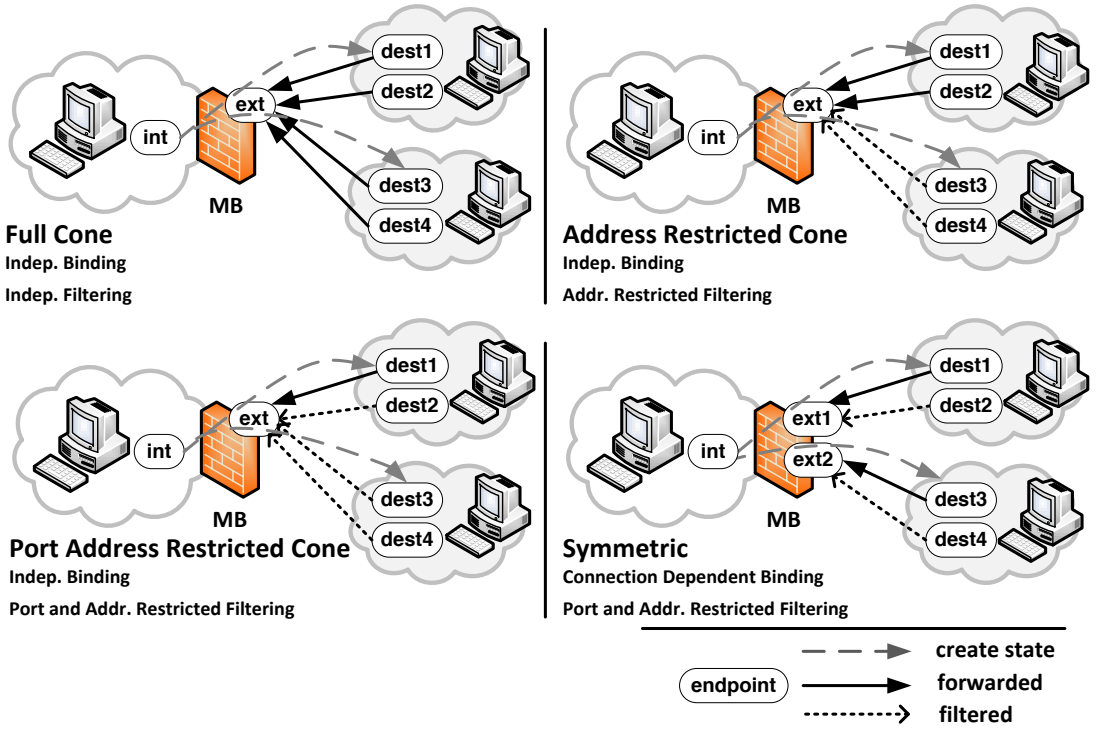

Fig. 3.2: Network Address Translation classification

This means the NAT has to maintain an additional entry for each connection, allowing it to consider the source address when filtering incoming packets. Thus, only packets coming from the same host the initial packet has been sent to ($dest1$ and $dest2$) can use the existing mapping. As a result, an internal host cannot be reached from arbitrary hosts in the Internet, which can be seen as a security plus.

Port Address Restricted Cone NAT

Again, endpoint independent binding is used for a Port Address Restricted Cone NAT. In addition to the destination address used by an Address Restricted Cone NAT, a Port Address Restricted Cone NAT also remembers the destination port of the outgoing connection. Now the external host B can only use the mapping from the port the initial packet was sent to ($dest1$). This extends the security consideration of the Address Restricted Cone NAT since the NAT acts like a stateful firewall and only forwards packets that are related to the ones that were sent by the internal host.

Symmetric NAT

Other than the three categories described above, a Symmetric NAT uses a connection dependent binding that assigns an external mappings dependent on the destination ($ext1$ for $dest1$ and $ext2$ for $dest3$). Dependent on the actual implementation, the inability to predict external ports with Symmetric NAT may lead to massive problems regarding the communication across NAT (NAT Traversal) since for every new connection, independent of the source or destination transport address, a new random mapping is assigned. Symmetric NATs usually behave like Port Address Restricted NATs when it comes to filtering.

3.3.4 Behavior of Large Scale NATs

The last sections described NAT behavior in general. When looking at Large Scale NATs, there are many additional behavioral issues that have to be considered. The first challenge is due to the fact that a LSN serves more than one customer. In a home network NAT shares an IP address among a number of trustworthy clients all having "equal rights". For example, it is not assumed that a client may behave evil and run a Denial of Service attack by randomly sending out TCP-SYN packets and thus creating too much state in the NAT. If a client does so, it is assumed that it doesn't matter that other clients are also affected. The same is true for legal issues. If a client behind NAT behaves illegally it is sufficient to identify the owner of the home network based on the public IP address. With a LSN this is not true anymore, since many homes will share the same public IP address. Thus, the operators of a LSN have to implement a logging mechanism to uniquely identify their customers. This of course uses up extra resources and further legal (privacy) issues may arise.

Another challenge is the possible impact of LSNs for customers. Since the LSN may have more than one public IP address, it is not clear when and how public IP addresses should be used when creating a new mapping in the LSN. Thus, some protocols may fail in such a scenario, e.g. when using different IP addresses for packets that logically belong to the same connection (layer 7 signaling). Furthermore, many Internet forums and message boards rely on IP addresses when blacklisting users (e.g. after a number of unsuccessful logins). Since public IP addresses will be shared among a number of customers, blacklisting with LSN is not possible anymore. Finally, providing services on hosts behind a LSN will be very hard for customers since they have no control over the NAT and are not able to enable port forwarding entries allowing to statically forward ports to private hosts.

The BEHAVE working group of the IETF standardized common requirements for LSNs in [114]. Besides general requirements addressing NAT behavior for UDP [4], TCP[59], ICMP [141] and the general problems that arise when sharing IP addresses [49], the behavioral requirements for LSNs are as follows.

Requirement 1 (R1) A Large Scale NAT can be seen as a shared resource and the available number of ports should be fairly shared by all customers. The NAT mapping table has to be kept as small as possible by assigning adequate timeouts for bindings. The limitation should be configurable by privileged users and should be adapted to the hardware of the LSN. One approach could be that the first packet of a customer instantly reserves a bin of ports, which also reduces the logging effort (see R2).

R2 R1 stated that short timeouts help to save resources. However, if a TCP timeout expires the port cannot be assigned to a new customer within a certain amount of time. It is recommended to wait at least 120 seconds, which is the Maximum Segment Lifetime in TCP [119]. This also depends on the filtering strategy the LSN implements. For address and port restricted filtering the port can be reused immediately.

R3 Since multiple customers may share the same external IP address, the NAT should store the following combination for uniquely identifying its customers: customerID; public IP address; timestamp.

R4 A LSN should use a paired address pooling [4], meaning that the external IP address always depends on the internal one. Protocols that use in-band signaling, such as SIP and FTP, establish multiple connections within one session. If the external IP addresses of these connections differ it is most likely that the protocol will fail.

R5 It is recommended that a LSN should implement an endpoint independent filtering strategy. This not only saves resources (see R1) by requiring less state in the NAT table, but also makes sure to have as little impact as possible on existing behavior-based NAT Traversal techniques such as hole punching [48].

R6 If R5 is fulfilled, most traversal methods will still work. However, for some scenarios, e.g. providing a service behind a NAT, it is necessary to directly forward certain ports to a specific destination. In the case of a LSN this is only possible if the customer is allowed to control at least a few ports of the LSN. However, this also implies a configuration protocol and raises security related questions.

3.4 Firewall Behavior

Section 2.4.3 already introduced stateless and stateful firewall behavior. The most important difference is that a stateless implementation simply looks up the firewall policy for every incoming packet and is not able to detect and handle related packets. Stateful firewalls maintain a table and keep track of established connections based on the state of the protocol. In this section we show how this behavior can be modeled using our notation. For both listings (algorithm 3.7 for stateless firewalls and algorithm 3.8 for stateful firewalls) we assume that the firewall is deployed between two networks and aims at protecting the internal network *green* from attacks coming from the external network *red*.

3.4.1 Stateless Firewalls

```
1 if allowed (policy := get_policy(p)) then
2 |   send(p, green);                    ▷ forward packet to internal network
3 else
4 |   drop_packet(p);                                          ▷ drop packet
5 |   log_incident(p);                                         ▷ logging entry
6 end if
```

Alg. 3.7: $Proc(red, p)$ for a stateless firewall

The individual processing steps for the function $Proc(external, p)$ are as follows:

1. The firewall receives packet p and looks up its policy database.
2. If the packet is allowed it is forwarded to the internal network (line 2).
3. If the policy states that the packet is not allowed it is dropped (line 4) and a logging entry is generated (line 5).

3.4.2 Stateful Firewalls

Algorithm 3.8 lists the notation for a stateful firewall that allows incoming packets for a specific port:

```
1  if state_related (state := get_state(p)) then
2  |  send(p, green);                          ▷ forward packet to internal network
3  else
4  |  if allowed (policy := get_policy(p)) then
5  |  |  allocate_new_state(p);                ▷ allocate new state for this connection
6  |  |  send(p, green);                       ▷ forward packet to internal network
7  |  else
8  |  |  drop_packet(p);                                          ▷ drop packet
9  |  |  log_incident(p);                                         ▷ logging entry
10 |  end if
11 end if
```

Alg. 3.8: $Proc(red, p)$ for a stateful firewall

1. The firewall receives packet p and checks if it belongs to an already existing (or "related") connection (line 1). If so, it is directly forwarded to the internal (green) network (line 2).
2. If the packet does not belong to an existing connection the firewall queries the policy database (line 3).
3. If the packet is allowed a new state entry is created (line 5) and the packet is forwarded to the internal network (line 6).
4. If the packet is not allowed it is dropped (line 8) and a logging entry is generated (line 9).

3.5 Application Layer Middleboxes and Proxies

Numerous middleboxes exist that operate above the transport layer and most of them act as a proxy providing application specific functionality. HTTP proxies cache information, filter advertisements and allow regulating the access to webpages. A SSL proxy can be used to intercept a secure connection in order to inspect the payload of these packets. The use of proxies can be configured by the client by explicitly entering the proxies address.

In any case a proxy will terminate the initial connection, process the packets according to its functionality and most likely establish a new connection to the original destination. In case of a caching HTTP proxy, the client could also be served directly from the cache without actually querying the destination. Therefore, modeling proxies is somehow straightforward. Packets are received, application specific processing rules are applied and a new connection is established. The following section serves as an example and shows the model of a Tor proxy.

3.5.1 Tor and Polipo

Tor[3] is a network for anonymizing TCP traffic. It protects its users by directing packets through multiple Tor servers and by onion routing [37]. To connect legacy applications

[3] https://www.torproject.org/

to Tor, a proxy operating between the transport and application layer is used. The Tor software packet comes with *Polipo*[4], a web-proxy supporting the SOCKS protocol [90]. Existing applications configure Polipo as a web-proxy and Polipo forwards their traffic to the Tor network via SOCKS.

```
1 if cached (state := get_state(p)) then
2 |  send(cached_answer, proxy_interface);          ▷ serve request from cache
3 else
4 |  send(p, tor_interface);                        ▷ forward request to Tor
5 end if
```

Alg. 3.9: $Proc(proxy\_interface, p)$ for the polipo proxy

1. A legacy application sends a packet p to the Polipo proxy which may serve the request from its cache (line 2).
2. If the request cannot be served from the cache the p is forwarded to the Tor application for further processing (line 4).

3.6 Summary and Key Findings

This chapter described middlebox behavior and introduced a processing model that allows describing the operation of arbitrary middleboxes. Based on this model we gave an insight into the behavior of Network Address Translation, firewalls and application specific proxies. The processing chain of the model is very generic and allows specifying arbitrary events in order to model individual processing steps of middleboxes. The granularity of the steps can be freely chosen dependent on the required level of detail. The processing model is the first step for answering the question if a thorough and structured analysis of middlebox behavior helps to improve the success rate of middlebox traversal techniques (Q1 according to section 1.1), which will be further examined in the following chapter.

Key Findings of this chapter
(and contributions according to section 1.2):

- **NAT, Large Scale NAT and firewalls are the most prominent examples of middleboxes violating the end-to-end argument.**
- **Middlebox behavior and especially NAT behavior is not standardized.**
- **In the state of the art there have been many efforts to characterize NAT behavior, but the impact of these classifications remains unclear.**

C3.1 **Our processing model helps to describe middlebox packet processing using arbitrary operations in order to understand the purpose and the operation of a specific middlebox.**

[4] http://www.pps.univ-paris-diderot.fr/~jch/software/polipo/

4. EXPERIMENTAL ANALYSIS OF MIDDLEBOX BEHAVIOR

4.1 Introduction

Dependent on the behavior and the purpose of introduction, a middlebox may have a significant impact on the quality of service, available bandwidth, latency or security of a connection between two endpoints. Moreover, if the middlebox implements firewall or NAT functionality it may even prevent a connection from being established. Unfortunately, middlebox behavior is not standardized and many vendors find different ways of implementing practically the same functionality. Some of them follow the specifications for standard protocols like TCP, others only implement the most important parts of it (e.g. ignoring the complete state diagram of TCP), leading to suboptimal behavior in many cases.

The goal of this chapter is to understand the behavior of middlebox implementations that are located on the edge of the network by experimentally measuring and analyzing them. Examples for such middleboxes are Large Scale NATs and proxies located at ISPs and Customer Premises Equipment (CPEs) such as home routers located in private networks. Most of these devices implement a non-standardized combination of NAT and firewall and only allow incoming packets as a response to outgoing packets. We argue that detecting and understanding middlebox behavior will help to find optimal ways for dealing with them. For example, instead of applying a traversal method using trial and error (Skype), our goal is to apply a customized version of a traversal algorithm that minimizes negative side-effects (e.g. breaking the end-to-end connectivity) of middleboxes.

After getting familiar with general measurement methods and tools in our lab, a public field test was conducted over a long period of time (approx. 4 years) to cover a large number of real middleboxes. During this period the measurement methodology has been adapted a few times to react to the most current findings. Today, the test completely runs in a web browser making it compatible with all major operating systems. The developed algorithms for measuring middlebox behavior also act as cornerstones for a new middlebox traversal framework that will be presented in chapter 6.

The following sections describe our experimental analysis in detail. First, section 4.2 presents an information model that is used throughout this thesis to refer to specific behavioral issues and to explain all further algorithms and results. Section 4.3 then introduces the individual algorithms for measuring middlebox behavior. Most of them are designed with the assumption of only having a very simple client that is only capable of sending basic UDP and TCP packets and does not require superuser privileges. The complete logic is therefore transferred to the server part that is running in the public Internet. After thoroughly analyzing a specific middlebox, an instance of the information model can be created that represents the middlebox and explains its behavior in detail. Section 4.4 presents a virtual testbed that dynamically creates an environment for emulating certain middlebox behaviors. Section 4.5 then focuses on related work, the design of the field test and on selected implementation issues. Finally, the results and the lessons learned are presented in section 4.6.

4.2 Information Model

In the last chapter a processing model was developed to express packet flows and individual processing steps of middleboxes. This section introduces an information model for describing behavioral characteristics and properties of middleboxes that are important for designing our measurement algorithms and for gaining knowledge about different implementations. The modeling was done in XML Schema [149] and the complete schema is listed in Appendix A. The model as presented in this section only contains its main elements. The individual fields and their possible values are then described in more detail when presenting our measurement algorithms.

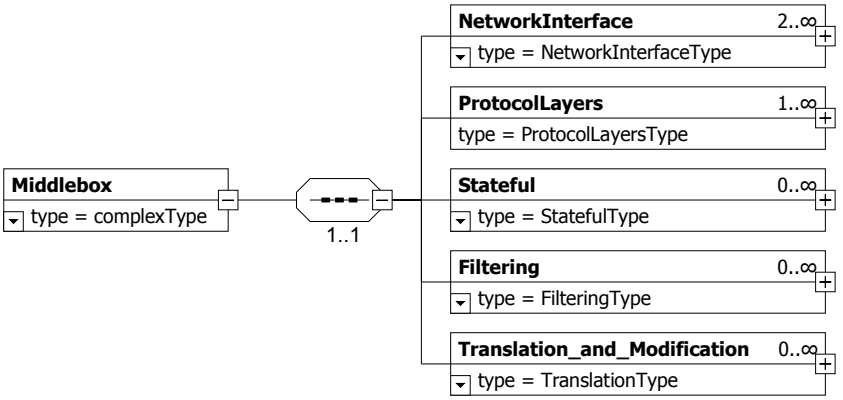

Fig. 4.1: Middlebox Information Model: Root elements

Figure 4.1 shows the root elements of our model. Each middlebox has a number of elements that may or may not be present (depicted by the cardinality of each element). There may be multiple network interfaces, each one carrying a specific name (e.g. *eth0*) (here we follow our definition from above and state that a middlebox has at least two network interfaces) and it may operate on different protocol layers. The middlebox may be stateful and it may filter and translate/modify packets on different layers. The following sections present the *Stateful*, *Filtering* and *Translation_and_Modification* elements in more detail. For the *NetworkInterface* and *ProtocolLayers* elements, please refer to Appendix A as these elements are rather straightforward.

4.2.1 Stateful Element

The stateful element is depicted in figure 4.2. Stateful middleboxes maintain a state table for storing and retrieving packet related information. For example, a NAT device stores the mapping between private and public addresses in its state table for retranslating incoming packets. The size of the state table is represented by an integer value (*TableSize*) and the *TableStrategy* field describes the behavior of a middlebox in case of a full table. In figure 4.2 the table strategy is represented by a *TableStrategyEnum* and is initialized with possible values for our thesis in the corresponding sections. Each protocol may maintain its own state table and the policy for each protocol may be different.

Each state in the state table has a certain expiration time. If no packet is seen for this amount of time, the entry is deleted. There may be multiple *StateTimer* entries for a stateful middlebox, each one representing a different timeout value. For example, a NAT device will have two timeout entries for the protocol UDP (UDP

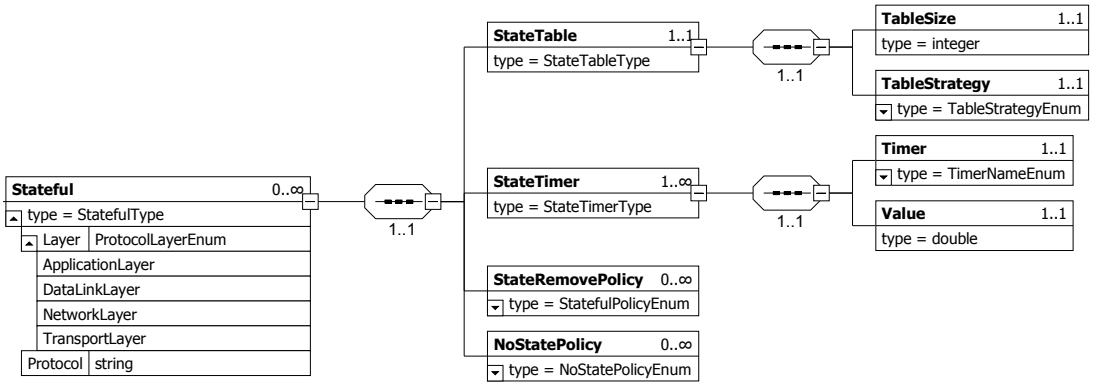

Fig. 4.2: Middlebox Information Model: Stateful element

timeout and stream timeout), as well as a number of timeouts for TCP (as shown in the following sections). The *StateRemovePolicy* field holds a description of policies that regulate when a state is removed. For example, some middleboxes remove the TCP state immediately after seeing a TCP-RST packet traveling from the internal network to the external one, thus the policy would be: "Remove State on TCP-RST out". The *NoStatePolicy* describes the reaction of a middlebox to packets that do not match an existing state, e.g. an incoming TCP-SYN could be silently dropped or answered with a TCP-RST.

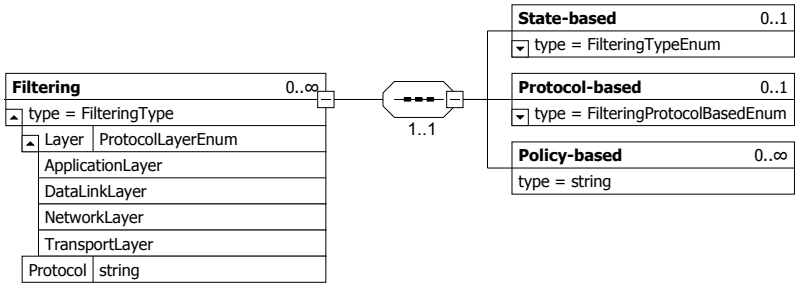

Fig. 4.3: Middlebox Information Model: Filtering element

4.2.2 Filtering Element

The filtering element as depicted in figure 4.3 describes the filtering of packets at the middlebox. The protocol that is affected, as well as the protocol layer is modeled as an attribute. On each layer and for each protocol multiple filtering mechanisms will apply. Filtering can be done based on certain fields of the state table (*State-based*) or simply based on a specific policy that is represented by a string value. For example, a NAT or firewall may as its default behavior filter incoming packets based on the address and port (thus address and port restricted). However, a user-specific policy could have been defined to drop all incoming UDP packets to port 5060. Additionally, middleboxes may filter specific content on the application layer which would also be reflected in the *Policy-based* field. *Protocol-based* filtering describes how the middlebox handles (unusual or unspecified) packet sequences of a certain protocol. For example, in TCP an incoming SYN packet (without any ACKs) as an answer to a previously

outgoing SYN packet is not a valid packet sequence. Some middleboxes might not filter this sequence, others do.

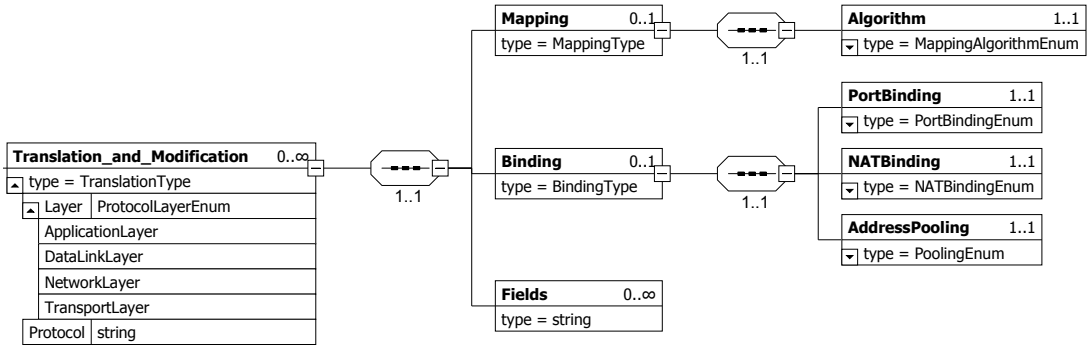

Fig. 4.4: Middlebox Information Model: Translation element

4.2.3 Translation Element

The translation and modification element as shown in figure 4.4 represents how packets are translated or modified along the way. General modifications (e.g. of the payload in case of an ALG) can be represented by using the *Fields* field. In case of address translation the *Binding* and *Mapping* fields allow specifying the translation operation in more detail. Binding is further divided into *PortBinding* and *NATBinding* and the mapping algorithm describes the way a middlebox assigns external addresses to new connections.

4.2.4 Reference Example of a Middlebox Instance

The following listing shows an example of a middlebox instance implementing Network Address Translation. Two network interfaces configured with IPv4 addresses are available and the middlebox implements a stateful filtering and translation behavior. Please note that the example was kept short for the sake of simplicity. A real instance would also include additional fields for the IP and TCP protocol (including additional timers), as well as a more detailed network configuration.

```
<?xml version = "1.0" encoding = "utf-8"?>
<n:Middlebox xmlns:n="http://example.org/MiddleboxSchema" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance" xsi:schemaLocation="http://example.org/MiddleboxSchema"
    MiddleboxType="referenceNAT">

  <NetworkInterface NetworkInterfaceName="lan">
    <ipv4>
      <IPAddress>192.168.1.1</IPAddress>
      ...
    </ipv4>
  </NetworkInterface>
  <NetworkInterface NetworkInterfaceName="wan">
  ...
  </NetworkInterface>
  ...
  <Stateful Layer="TransportLayer" Protocol="UDP">
    <StateTable>
      <TableSize>8000</TableSize>
```

```
      <TableStrategy>Block</TableStrategy>
    </StateTable>
  ...
    <StateTimer>
      <Timer>UDP</Timer>
      <value>30</value>
    </StateTimer>
        ...
  </Stateful>
  <Filtering Layer="TransportLayer" Protocol="UDP">
    <State_based>Address and Port Restricted</State_based>
  </Filtering>
  <Translation_and_Modification Layer="TransportLayer" Protocol="UDP">
    <Mapping>
      <Algorithm>Increment</Algorithm>
    </Mapping>
    <Binding>
      <PortBinding>PortPreservation</PortBinding>
      <NATBinding>EndpointIndependent</NATBinding>
    </Binding>
  </Translation_and_Modification>
</n:Middlebox>
```

List. 4.1: Reference example of a middlebox implementing NAT

4.3 Detailed Description of our Measurement Algorithms

This section develops algorithms with the goal of analyzing different parts of the information model. All of them work in a way that the middlebox to be tested is located between a test client and a test server that exchange a number of packets (according to the algorithms). The individual algorithms can be split into four categories as listed below (table 4.1).

Sec.	Test	Information Model
Behavior Analysis		
4.3.1	**Binding Analysis**	*Translation_and_Modification:Binding*
	This tests aims to determine the binding strategy of a translating middlebox.	
	Mapping Analysis	*Translation_and_Modification:Mapping*
	Detects the mapping strategy for new ports.	
	Filtering Analysis	*Filtering:State-based*
	Determines the filtering strategy (which inbound packets are allowed).	
	Timeout Analysis	*Stateful:StateTimer*
	Examines timeouts for different protocols.	
	ICMP Analysis	*Filtering:Protocol-based*
	How the middlebox handles ICMP messages.	

	Mapping Table Analysis	*Stateful:StateTable*
	Size and strategy analysis of the mapping state table.	
Traversal Analysis		
4.3.2	**UDP Hole Punching high**	*Stateful:Policy*
	Triggers an ICMP port unreachable after an outgoing UDP HP packet.	
	UDP Hole Punching low	*Stateful:Policy*
	Triggers an ICMP TTL exceeded after an outgoing UDP HP packet.	
	UDP Hole Punching silent	*Stateful:Policy*
	Triggers no packet after an outgoing UDP HP packet.	
	TCP Hole Punching high	*Stateful:Policy*
	Triggers a TCP-RST packet after an outgoing TCP-SYN.	
	TCP Hole Punching low	*Stateful:Policy*
	Triggers an ICMP TTL exceeded after an outgoing TCP-SYN.	
	TCP Hole Punching silent	*Stateful:Policy*
	Triggers no packet after an outgoing TCP-SYN.	
Additional Measurements		
4.3.3	**Universal Plug and Play**	*ProtocolLayers:7:UPnP*
	Gathers as much information as possible about the middlebox.	
	SIP ALG	*ProtocolLayers:7:SIP*
	Tests if the middlebox implements a SIP ALG.	
	FTP ALG	*ProtocolLayers:7:FTP*
	Tests for a FTP ALG and if it can be (mis)used for traversal.	
	Image Proxy	*Translation_and_Modification:7:JPEG*
	Tests if the middlebox modifies JPEG images in the payload.	
	SCTP Support	*ProtocolLayers:4:SCTP*
	Determines if the middlebox supports the SCTP protocol.	
	IPv6 Analysis	*ProtocolLayers:3:IPv6*
	Examines if the middlebox supports IPv6.	
Topology Measurements		
4.3.4	**Topology Detection**	*Stateful:StateTimer*
	Detects deployed middleboxes and their properties along the path.	

Tab. 4.1: Overview of the individual experiments

4.3.1 Behavior Measurements

The first measurement category tests the behavior of middleboxes according to our information model. Since many middlebox vendors implement the same functionality in a different way, our goal is to categorize different behaviors to understand side effects of incorrect implementations and to design middlebox traversal methods that support as many implementations as possible.

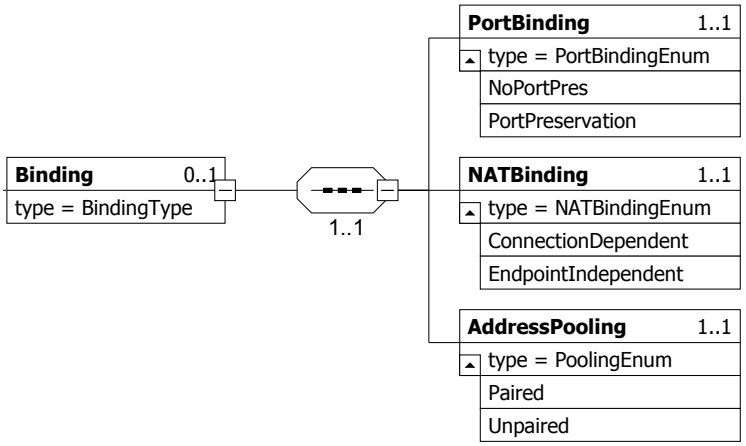

Fig. 4.5: Middlebox Information Model: Binding element

Binding Measurements

Our first experiment analyzes the binding behavior of a translating middlebox. As presented in section 3.3, binding describes the strategy a middlebox uses to assign a public transport address to a new connection. According to the information model, this experiment analyzes the *PortBinding* and *NATBinding* part of the *Translation_and_Modification* element for the protocols UDP and TCP as depicted in figure 4.5.

```
1 sourcePort = randomPort();
2 sendUDP/TCPpacket to TestServer1 from sourcePort;
3 TransportAddress1 = receiveFromServer();
4 sendUDP/TCPpacket to TestServer2 from sourcePort;
5 TransportAddress2 = receiveFromServer();
6 compare (TransportAddress1, TransportAddress2, localAddress);
```

Alg. 4.1: Binding pseudo-code as seen by the client

Algorithm 4.1 shows the individual steps of the binding measurement algorithm. The client sends two packets from the same local source address and port to different external servers. The server then returns the public transport address within this connection. By comparing its original source address with an external one the client knows if it is behind a middlebox that manipulates transport addresses (NAT functionality). This works for IPv4 NAT, but in an IPv6 NAT environment, the client side IP address might already be a public one. Furthermore, the client is able to see if the NAT preserves ports (see section 3.3). The possible results by comparing the external mappings are the following:

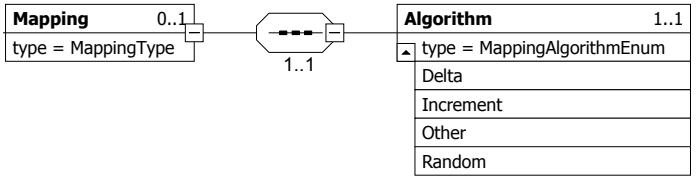

Fig. 4.6: Mapping element

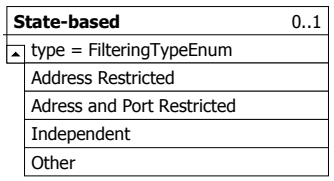

Fig. 4.7: Filtering element

IP addresses and ports are the same: If both external IP addresses are the same the middlebox is either only using one external IP address or it applies a paired address pooling behavior (see section 3.3.4). Since both external source ports match, the binding behavior can be considered as endpoint independent. We expect that many standard home routers that share a single IP address among a small number of devices fall into this category. As long as the external mapping is only dependent on the source port, this category allows querying public mappings by contacting an external server such as STUN.

IP addresses are different, but ports match: In the second case the external source ports match, but the IP addresses are different. In this scenario the deployed middlebox uses multiple global IP addresses. Most likely this would be a middlebox at the ISP implementing load-balancing and violating the recommendation of a paired address pooling behavior.

IP addresses match, ports are different: If the external IP addresses of both outgoing connections match but the ports are different we are dealing with a connection dependent binding strategy. This means, the external endpoint is not only dependent on the source, but also on the destination. In this case port prediction is rather difficult and depends on the port allocation algorithm of the middlebox. We expect this behavior to be seen at a few home routers and in many larger implementations following the symmetric NAT approach.

IP addresses and ports are different: This case is a combination of the last two cases. The IP addresses as well as the external source ports of the mapping are different. This means the middlebox implements a connection dependent mapping and it uses load-balancing or some other mechanism violating the paired address pooling behavior.

Mapping and State-based Filtering Measurements

The goal of the conducted mapping tests is to find out how a translating middlebox allocates resources for outgoing connections. This may help to find allocation patterns (*Translation_and_Modification:Mapping:Algorithm*) that allow predicting external endpoints. More specifically, with NAT we are interested in the allocation of IP addresses and ports to new mappings, referring to the *Mapping* part of the *Translation_and_Modification* element as shown in figure 4.6. Additionally, by running the STUN [130] algorithm, we will also learn about the filtering behavior which is mapped to the *Filtering:State-based* element as depicted in figure 4.7.

```
1 choose arbitrary starting source port;
2 while number of tests not reached do
3 |   sendUDPpacket to TestServer;
4 |   clients public TransportAddress = receiveFromServer();
5 |   rememberMapping;
6 |   sourcePort++;
7 end while
8 storeMappings();
9 runSTUN();
```

Alg. 4.2: Mapping and filtering pseudo-code for UDP as seen by the client

The pseudo-code of the mapping test for UDP is depicted in algorithm 4.2. For TCP a new connection is established for every source port. The whole test is repeated a few times using different starting source ports to gather as much information as possible. After collecting the data a port prediction algorithm tries to find a logical schema how the NAT allocates public mappings. More details on port prediction can be found in section 5.2.2. As a result, the client gets a paired list of combinations of internal and external mappings. By comparing these mappings the results are as follows:

IP addresses and ports are the same: If the internal and external IP address and the ports match there is no translating middlebox involved. In combination with the traversal tests it should be tested if there's a stateful firewall involved.

IP addresses are different, but ports match: In the second case the internal and external source ports all match, but the IP addresses are different. We expect this to be a very common scenario, e.g. a home router, where the middlebox only uses one external IP address. Since all ports match the middlebox either tries to preserve ports or, in case of IPv6 NAT, the middlebox only operates on the network layer. This can be checked by comparing the checksums of the internal and external packet, since NAT66 [159] uses a checksum neutral mapping.

IP addresses match, ports are different: In this scenario the middlebox only translates the source ports, but not the IP addresses. Here we assume that the source IP address is already coming from the public range in order to make sure no NAT device is involved. We don't expect to discover this scenario in our test since stateful NATs have the purpose of translating at least IP addresses and firewalls do not change port numbers.

IP addresses and ports are different: We expect this behavior to be very common for home routers that do not preserve source ports on outgoing mappings. One reason for not preserving ports is that the external port is not available anymore. Thus, we have to check multiple port ranges. In order to design a working port prediction algorithm we aim to carefully look at the actual assignment strategy. Some of the middleboxes may start at a pre-defined external source port and *increment* external ports by a fixed delta, while others may choose a *random* external port. Please refer to the information model for a complete list: *Translation_and_Modification:Mapping:Algorithm*.

In addition to the described steps the test also queries a STUN server [133] to verify the results. STUN is helpful to determine the state-based filtering strategy of the

middlebox (see section 3.3.2 and *Filtering:State-based*) to find out which external clients are allowed to access the mapping. This becomes important when considering different service categories for middlebox traversal as described in section 5.3.

Mapping Timeouts

Since a stateful middlebox only has limited resources, it needs to carefully manage the number of entries that are held in its mapping table. More specifically, the middlebox needs to delete the mapping of connections that have been terminated or idle for a certain amount of time. Furthermore, dropping idle connections after a certain amount of time also helps to be less vulnerable to Denial of Service Attacks.

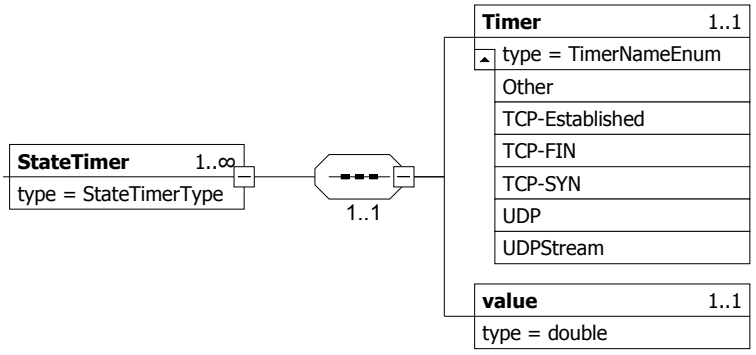

Fig. 4.8: Timeout analysis: State timer of the Information Model

Managing mapping entries is done by assigning timers to every outgoing connection (see figure 4.8). If the timer expires, the mapping is deleted. If a new packet is seen on the connection, the timer is reset. For stateful protocols such as TCP, middleboxes usually assign timeouts for established connections that are longer than the ones for UDP. In [60] the authors measured the timeouts for different phases of a TCP connection (SYN-sent, established, timed-wait and closed) and concluded that "applications should not rely on idle connections being held open for more than a few minutes".

For UDP the situation is slightly different since it is a connectionless protocol. The middlebox assigns a timer to each entry and drops the mapping once the timer expires. In order to keep a UDP mapping open, the client behind the middlebox will have to refresh the connection by sending another UDP packet to the same destination. Determining a reasonable refresh interval is one goal of our UDP timeout test. Especially for mobile clients the refresh interval should not be too short in order to save bandwidth, as well as to reduce possible battery drain. Additionally, many middleboxes have a multi-leveled way of handling UDP timeouts. The more the packets are being sent over the same connection, the larger the timeout. This timeout is also referred to as the *UDP stream timeout*.

The timeout test considers UDP timeouts, as well as TCP timeouts. However, since testing all TCP timeouts may take up to several hours, we only test for timeouts in the *established* state with a threshold of five minutes. The reason for this is that only in the established state the timeout is relevant for actual protocols. For example, when registering for a service such as SIP via TCP, the client is waiting for messages coming from the server. The TCP connection is already fully established, but the arrival time of incoming messages is not predictable and it may take several hours until the server actually sends data to the client.

For the UDP tests our tester creates several mappings in the middlebox by sending out single UDP packets with increasing source ports. Within the payload of the UDP packet the client specifies the timeout to be tested (e.g. 10 seconds). The server part receives the packets, waits for the defined period of time and sends a UDP packet back to the client. If the client receives the answer it assumes that the mapping is still alive. The test has to be repeated multiple times since UDP packets may get lost on their way to the server. Sending multiple UDP packets at the same time to make sure at least one of them reaches the server doesn't work since the client would run into the UDP stream timeout. This is tested separately by using different ports.

ICMP messages

The Internet Control Message Protocol (ICMP) [118] is used to inform hosts about possible errors, such as expired TTLs and unreachable destinations. A middlebox has to implement the ICMP protocol in order to translate and forward these packets between the realms. Furthermore, if the middlebox changes addresses on the network or transport layer, it also has to take care of the payload of certain ICMP messages. This is because many messages such as *ICMP TTL expired*, which is sent by the hop where the time to live field of an IP packet expires, include the original IP packet and parts of the TCP/UDP packet in their payload [118]. The hop sending the expired packet only sees the public transport address of the IP packet, whereas the host that originally sent the packet expects to see the private one. Therefore, the middlebox needs to translate transport addresses on outgoing ICMP packets, as well as transport addresses on incoming packets.

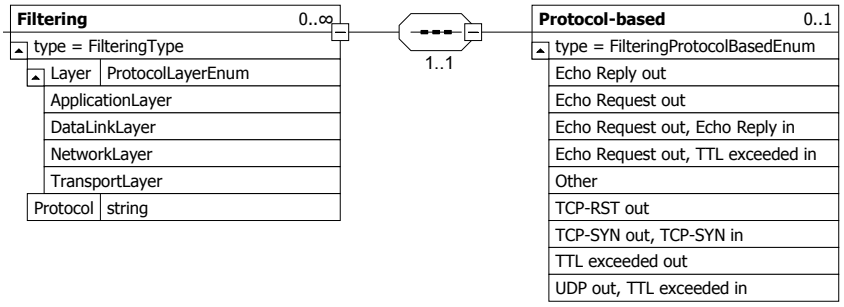

Fig. 4.9: Filtering analysis: Protocol-based filtering

Our test also checks how middleboxes handle ICMP packets. In addition to regular traffic (outgoing UDP triggers incoming ICMP TTL expired), we also test for packet sequences that are normally not seen (e.g outgoing ICMP without related incoming packet). Furthermore, we also test if the payload of the ICMP packets is relevant for any forwarding decision or if it is possible to manipulate it. This is useful for sending signaling messages via ICMP and can be used by our autonomous middlebox traversal algorithm as presented in section 6.5.2. We distinguish between two types of tests, incoming and outgoing, as follows:

The goal of the **incoming tests** is to check whether the middlebox handles incoming ICMP packets that are sent as a response to outgoing packets. It also checks if the payload of the packets is translated correctly. For all combinations we check if it is allowed to manipulate the payload of the incoming packet, e.g. add a few bytes at the end or replace certain fields of the embedded headers.

- *Echo Request out, TTL exceeded in* checks if the middlebox allows an incoming ICMP TTL exceeded as a response to an outgoing echo request that established a state in the middlebox.
- *UDP out, TTL exceeded in* checks for a valid packet sequence.
- *Echo request out, echo reply in* simply checks if an internal host is allowed to ping an external one.

The goal of the **outgoing tests** is to check if the middlebox allows sending out ICMP packets without having a state for them. Again we also check for different payloads.

- *TTL exceeded out* checks if the middlebox allows ICMP TTL exceeded packets without having received any related UDP packets.
- *Echo request out* checks if the middlebox allows outgoing ICMP echo requests.
- *Echo reply out* checks if it is possible to send an echo reply without receiving any requests.

State Table Analyzer

Stateful middleboxes need to allocate resources for every entry in the mapping table. Therefore, the number of simultaneous connections is limited. This is in particular true for resource restricted devices such as home routers that usually only come with a few megabytes of RAM. The state table analyzer tests tries to approach the maximum number of simultaneous connections that a middlebox is able to handle. Additionally,

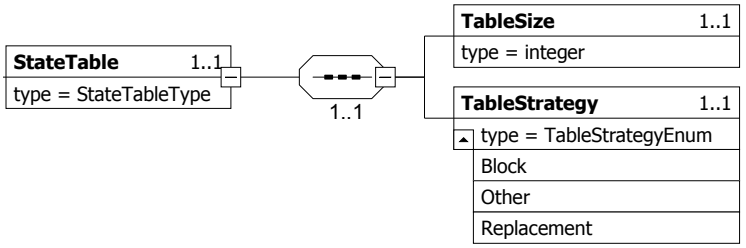

Fig. 4.10: State table element of the Information Model

it tests the strategy the middlebox uses for replacing existing entries. For example, if the mapping table is full, does the middlebox still allow additional connections? This could be for example done by overwriting existing ones using a strategy such as FIFO (first in first out) or LRU (least recently used). The problem with this test is the large number of connections that have to be established at the same time, which requires a careful management of processes and sockets when implementing it. Additionally, once a new connection is established the old ones have to be tested if they are still alive. As a result we can determine the maximum number of connections the middlebox supports (*StateTable:TableSize*), as well as the behavior when approaching this limit: *StateTable:TableStrategy*, both according to figure 4.10. The algorithm for analyzing the state table is as follows:

```
1  while 1 do
2      if newConnectionSuccessful then
3          checkAllOldConnections;
4          if oldWorking then
5              continue;
6          else
7              replacementStrategy;
8              break;
9          end if
10     else
11         if oldWorking then
12             reachedLimit;
13             break;
14         else
15             reached limit with errors;
16             break;
17         end if
18     end if
19 end while
```

Alg. 4.3: NAT mapping table pseudo-code

4.3.2 Behavior-based Traversal Analysis

There are numerous approaches for middlebox traversal (that will in detail be described in the following chapter), but only behavior-based techniques are independent from an active support of the middlebox. The most common behavior-based approach is hole punching. With hole punching a peer behind a middlebox first sends a packet towards the other peer, thus establishing a mapping in the middlebox (or punching a hole, thus the name). The other peer then tries to send a reply packet that matches the so-created state. The success rate depends on the implementation of the middlebox, more specifically if the middlebox allows certain incoming packets as a response to outgoing ones, if the middlebox filters certain packets in general and if the middlebox removes a state if it sees specific packets. Thus, our traversal analysis focuses on the *Stateful:Policy* (both *StateRemovePolicy* and *NoStatePolicy*) and on the *Filtering:Protocol-based* elements of the information model.

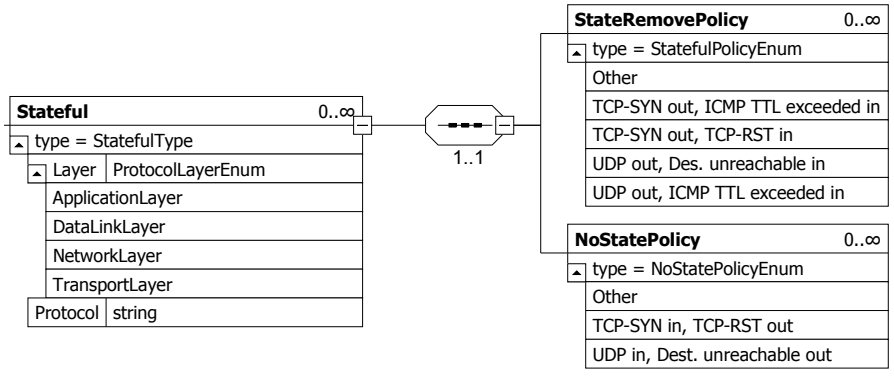

Fig. 4.11: Policy-based state maintenance

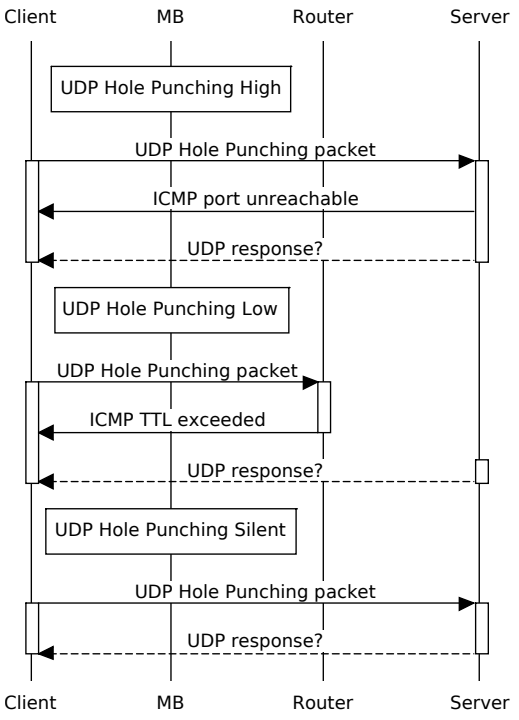

Fig. 4.12: UDP traversal

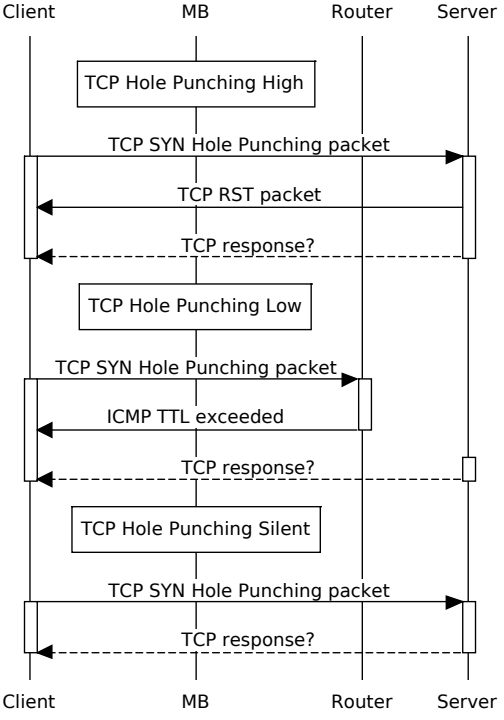

Fig. 4.13: TCP traversal

Figure 4.12 shows our approach for UDP hole punching. In the first experiment, *UDP Hole Punching High*, the test server replies with an ICMP port unreachable packet before testing for the actual UDP traversal. With *UDP Hole Punching Low*, the TTL of the initial UDP packet expires at an intermediate router, which sends back an ICMP TTL exceeded packet. The test server then tries to send the matching UDP packet in order to reach the client. Finally, *UDP Hole Punching Silent* tests if UDP traversal is possible by predicting the external port of the mapping.

[60] proposes STUNT, a technique that requires the NAT to forward an incoming TCP-SYN packet as a response to an outgoing one. Figure 4.13 shows three variants of this approach. With *TCP Hole Punching Silent* we test for the general capability of accepting a SYN-in-SYN-out sequence, which corresponds to the *Filtering:Protocol-based* field of our model. With *TCP Hole Punching High* the server sends a TCP-RST packet as a response to the initial TCP-SYN and checks if the state still exists or if a stateful policy exists (*Stateful:StateRemovePolicy*). *TCP Hole Punching Low* sets the TTL value of the initial IP packet carrying the TCP-SYN to a lower value, thus generating an ICMP TTL exceeded packet sent by an intermediate router.

4.3.3 Additional Measurements

Middleboxes may implement numerous additional features and may support a large number of protocols on the application layer. This section focuses on the measurement of protocols and features that are promising to be widespread.

UPnP

The Universal Plug and Play (UPnP) protocol suite[1] allows querying UPnP capable devices by sending multicast messages to a local network. If a middlebox implements UPnP it responds with useful information such as the model and the public IP address. *ProtocolLayers:7:UPnP* describes UPnP specific fields and we are especially interested in the following ones:

- UPnPManufacturer: returns the manufacturer of the middlebox
- UPnPModel: returns the exact model, e.g. DI-504
- UPnPFriendlyName: returns the friendly name, e.g. "myHomeRouter"
- UPnPExternalIPaddress: lists the current external IP address
- UPnPlasterror: returns the code of the last error (implementation specific)
- UPnPuptime: returns the uptime of the middlebox

Application Layer Gateways

Some middleboxes implementing translation mechanisms also operate on the application layer for some protocols. The Session Initiation Protocol (SIP) [131] together with the Session Description Protocol (SDP) [67] is mainly used for Voice over IP and causes problems with middleboxes since it embeds domain specific IP addresses and ports in its payload. Many middlebox thus implement an Application Layer Gateway (ALG) that understands the protocol and translates the addresses according to the realm. The same is true for the File Transfer Protocol (FTP) [120]. In active mode, the client signalizes the server an IP address and port where it awaits the actual data connection. If the addresses are private, a connection will not be possible. A FTP ALG translates private addresses to public ones and takes care of establishing a mapping in the state table of the middlebox. Our experiments test for both of these ALGs and the results are listed in the *ProtocolLayer:7:FTP* and *ProtocolLayer:7:SIP* fields.

Furthermore, we also try to misuse the FTP ALG for traversal: A fake FTP client sends out a packet carrying the PORT command of an actual FTP packet and specifies an IP address and port where our test service is waiting for incoming packets. If the ALG actually establishes a mapping for this address, an external peer is able to connect to it using an arbitrary TCP connection. This behavior is documented in the *Translation_and_Modification:7:FTP* field.

Image Proxy

This experiment tests for a proxy that manipulates JPEG images by resizing them. Mobile operators often deploy such proxies to reduce traffic within their networks. We download multiple images of different sizes via HTTP and check for modifications.

Additional Protocols

Newly developed protocols are usually not understood by most middleboxes, which hinders their deployment. The Stream Control Transmission Protocol (SCTP) [146] is an alternative transport layer protocol that most middleboxes are not aware of. Due to the extended checksum in the SCTP header and due to its multi-homed approach, the translation of SCTP is not as straightforward as the translation of TCP and UDP [145]. We test for the ability of middleboxes to translate the SCTP protocol by establishing an

[1] http://upnp.org/sdcps-and-certification/standards/

association towards the Internet. Additionally, we try to establish an IPv6 connection to get more information about the deployment of IPv6, as well as the support of IPv6 at different middleboxes. All results are reflected in the corresponding *ProtocolLayers* element of the information model.

4.3.4 Topology Measurements

The goal of our topology measurements is to detect stateful middleboxes along the path between two peers. In our scenario a test server located in the public Internet acts as a reference point that arbitrary clients use to reveal their network topology towards the server. Additionally, we gather as much information as possible from intermediate hops, such as their individual timeouts.

There are already a few approaches to middlebox detection in the state of the art. First, the STUN protocol [133] allows a peer to retrieve its IP address and port as seen by a STUN server that is usually deployed in the public Internet. Thus, the topology is hidden and only the addresses of the outermost middlebox are detected. However, if a STUN server would be deployed in every intermediate network, it would be possible to reveal the complete topology by querying one after another. STUN's control extension [132] defines an interface for middleboxes allowing to query them directly. The same is true for the NSIS Signaling Layer Protocol [147] and the Port Control Protocol (PCP) [162]. However, if intermediate hops don't implement the extension, it is not possible to detect the topology.

A naive approach to detect stateful middleboxes without requiring active support is to send out UDP messages with different TTLs. As soon as the packets expire, intermediate routers send ICMP TTL exceeded packets including the headers of the initial UDP packet. By comparing the IP addresses a client would be able to detect translating middleboxes. However, today's middleboxes also retranslate the headers in the ICMP payload, which makes this approach useless.

Approach

Our approach as initially developed in [163] and depicted in figure 4.14 conducts multiple measurements from the test server towards the client. In order to reach the client from the public Internet, the first step is to establish a mapping in all intermediate nodes. Thus, a client starts the measurements by sending packets to the test server. Once the server has received these packets, it counts the number of hops between the server and the client by conducting a traceroute[2] measurement. To get reliable results a stable path (paths may vary due to load-balancing) is required. Thus, the number of hops between the client and the server should be as low as possible. This can be reached by geographically distributing multiple test servers in the public Internet.

Once the server has detected the initial topology the client successively removes each mapping starting from the one closest to it. After removing a mapping the server repeats the traceroute to the same destination and compares the number of hops with the initial result. In figure 4.14 the mapping of MB1 is removed first and the following traceroute from the server to the client is blocked by the middlebox since no state exists anymore. Thus, the result of the traceroute reveals one hop less than the initial hop count, which is used by the server as an indicator that there is a stateful middlebox at hop 1. In the second step the client directs Router1 to remove its mapping. Since it is no stateful middlebox the following traceroute still reaches MB1 with the same hop

[2] http://linux.die.net/man/8/traceroute

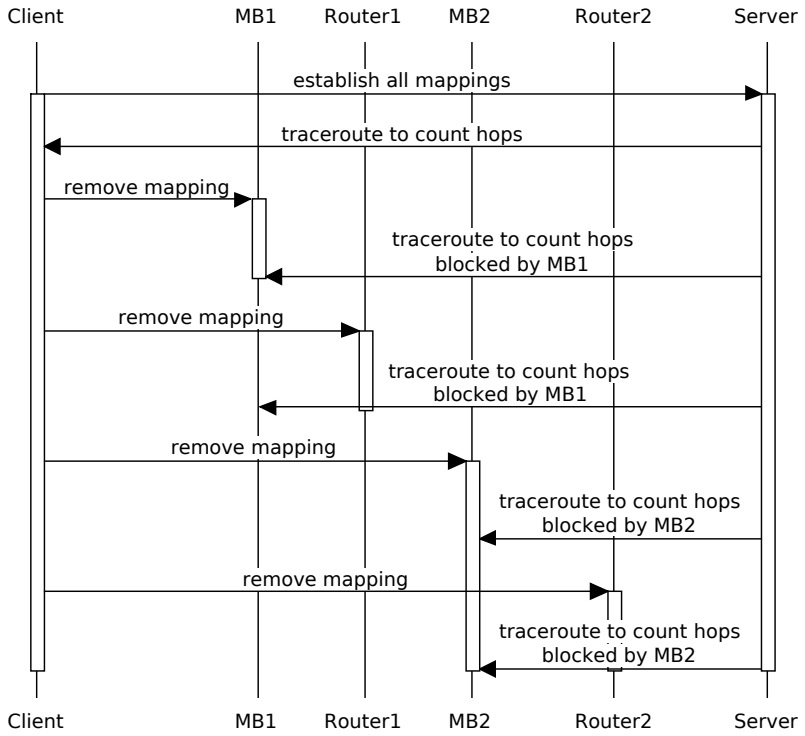

Fig. 4.14: Illustration of the topology detection approach

count as before. Thus, hop 2 does not implement stateful filtering. The pseudo-code of our algorithm is shown in the following listing.

```
1 establish mapping in all middleboxes along the path;
2 start from n=1, increment n by 1 for each run and
3 repeat
4 |   remove mapping of all hops from 1 to n as seen by the client;
5 |   count number of hops from server to client;
6 until n reaches number of hops;
```

Alg. 4.4: Middlebox topology detection algorithm

The server keeps track of all traceroute measurements and stores the number of hops that were successfully traversed for each timeout value. For each such combination, the server also knows the value of the client's current counter (variable n in the above pseudo-code) representing the hop for which the client just removed the mapping. As a result, the server is able to assemble a table and derive the number of stateful middleboxes from it. A complete result table together with a real topology is shown in section 4.6.7 when presenting our topology results.

Remove Mapping

Our algorithm requires the capability of a client to remove a mapping in an intermediate node. Without a protocol that actively controls middleboxes this is not a trivial task. However, when looking at the *Stateful* element of our information model, there are two possibilities to remove an established state: First, a *Stateful:StateTimer*

is assigned to each mapping and once it expires, the mapping is deleted. Second, a *Stateful:StateRemovePolicy* might exist that removes a mapping based on a sequence of packets.

In case of TCP, a client would establish a connection to the test server, thus creating a mapping in each intermediate node. As long as the connection is in the established state, the mappings are kept in the middleboxes (assuming the TCP-established timer is large enough). As soon as a middlebox sees a TCP-RST packet it might (according to its *Stateful:StateRemovePolicy*) remove the mapping immediately. However, if the client simply terminates the existing connection, the TCP-RST packet is sent to the test server via all intermediate nodes, which remove their mappings. Thus, in order to remove the mappings step by step, the initial connection has to be kept open and a TCP-RST packet has to be sent in a way that it only reaches a specific hop. This can be done by setting the TTL value of the IP packet to an appropriate value. The disadvantage of this approach is that sending plain TCP-RST packets with a specific TTL requires RAW sockets and therefore superuser privileges.

For UDP no such RST packet exists. Thus, waiting for a timeout is the only possibility to remove a mapping from a stateful middlebox. The following listing shows our algorithm for UDP:

```
1 foreach hop from 1 to n do do
2 |   establish mapping: send UDP packet from client to server;
3 |   server waits for UDP Timeout and sends keep-alive packets with
  |   TTL = #Hops − n;
4 |   traceroute from server to client;
5 end foreach
```

Alg. 4.5: Middlebox topology detection algorithm for UDP

In order to make sure only the timeout of the innermost middlebox (the one to be tested) expires, the server has to send keep-alive packets to all the other hops. This can be done by setting the TTL value of the corresponding IP packet in a way that the keep-alive packet expires right before it reaches the middlebox to be tested. Since the timeout value is not known beforehand and may vary from middlebox to middlebox, a large number of tests using different timeout values should be run in parallel.

Additional Results

The topology measurement not only reveals information about stateful middleboxes on the path, it also delivers additional results. First, when using the UDP approach the timeouts of the individual hops are not known in advance, resulting in many parallel tests with increasing timeout values. When running sufficiently enough such tests, timeouts of each hop on the path (*Stateful:StateTimer*) can be detected. Second, port allocation patterns of the outermost middlebox can be detected by monitoring the source IP addresses and ports of packets coming from the client. Third, by comparing hop counts, the stability of a path can be examined. Finally, it can be measured if providers block outgoing ICMP messages, which is a real problem for getting accurate results. The following listing gives an overview of additional results that can be gathered by the algorithm:

- By running parallel tests with increasing timeouts, *Stateful:StateTimer* can be detected for each hop.

- By monitoring source addresses of incoming packets, binding and port allocation patterns can be detected: *Translation_and_Modification:Binding*.
- By comparing hop counts, the path stability can be monitored. Unstable hop counts are strong indicators for load-balancing.
- By comparing TCP and UDP results, filtered ICMP TTL exceeded packets can be detected: *Filtering:Protocol-based*.

4.3.5 Active Monitoring of Middlebox Parameters

All approaches that we have described so far are based on measurements that don't require cooperation of the middlebox. With our Lightweight Information Export tool LinEx [106] we developed a tool in the context of home networking that can also be used for actively collecting and exporting status information from the middlebox to a collection agent. LinEx runs on embedded Linux routers and is able to export information from system configuration files, values read from the proc-filesystem, and the output of command-line tools. The export can be either done by tools such as *scp*[3] or email, or it can continuously export information via the IP Flow Information Export (IPFIX) Protocol [25].

In the following we describe which network information can be typically found in the proc-filesystem, in configuration and status files and in the output of command-line tools of an OpenWRT[4] enabled middlebox.

proc-Filesystem

The virtual Linux processing filesystem (*/proc*) holds process information generated by the kernel, such as state parameters related to the entire system (e.g. memory information in */proc/meminfo*), to specific devices (*/proc/devices*), or to running processes (*/proc/PID*). All entries can be easily extracted using standard file operations (e.g. *fopen* or the command-line tool *cat*). Low-level information about network interfaces can be found in */proc/net* and */proc/sys/net* although it is usually more convenient to execute the command *ifconfig* which provides a summary of the relevant information. The connection tracking (conntrack) function of the netfilter kernel module tracks TCP connections and other IP packet flows traversing the router. A list of currently active connections and flows can be obtained from */proc/net/nf_conntrack*. Middlebox specific information such as timeout values (e.g. *nf_conntrack_udp_timeout*) and the maximum mapping table size *nf_conntrack_max* can be found at */proc/sys/net/netfilter*.

Configuration and Status Files

Just like on Linux PCs, configuration files are usually located in the */etc* directory while files with status information can be found in */var*. The availability depends on the installed services and applications. The name and location of a specific file may vary between different router firmwares.

On OpenWRT, configuration files are located in the */etc/config/* directory. As an example, the DHCP configuration is stored in */etc/config/dhcp*. This file contains the interfaces for which DHCP is enabled. The list of active DHCP leases in the file */var/dhcp.leases* gives an overview on currently connected devices, including their IP and MAC addresses. Interface and routing parameters are stored in */etc/config/network*, firewall settings are located in */etc/config/firewall*.

[3] http://linux.die.net/man/1/scp
[4] http://www.openwrt.org

Output of Command-line Tools

Some volatile information, such as interface statistics and the list of connected WLAN clients, cannot be obtained from a file, but requires the execution of a command-line tool. Appropriate tools are *ifconfig*, but *iptables -l* is also useful for determining the current firewall configuration.

4.4 Experiments in the Lab and Verification of our Algorithms

4.4.1 Test Setup

To validate our algorithms we set up a testbed that consists of a test server, a test client and an arbitrary number of middleboxes. The interfaces of the middleboxes are configured using increasing IP addresses and by changing the default gateway the client decides which middlebox to test. This setup is depicted in figure 4.15.

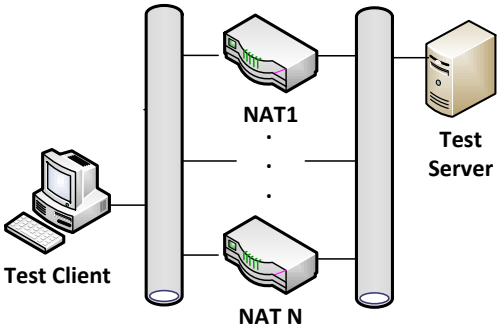

Fig. 4.15: Simplified testbed for our home routers

We bought a number of home routers, introduced them into our testbed and ran our algorithms to test their behavior. In this case the routers are treated as black boxes since their behavior is not known, documented or standardized. Thus, most behavioral algorithms cannot not be verified in this setup since the correct result of a test is not known beforehand. If an algorithm returns an output we can only assume that the middlebox behaves in a certain way, but the outcome of the algorithm could also be due to a wrong implementation.

4.4.2 Virtualized Testbed and Topology Generator

To actually verify the results of our algorithm we need to know the exact behavior of a middlebox before testing it. Since the behavior of existing middleboxes is not known, our goal is to emulate certain behavior aspects for middleboxes using legacy Linux machines. In section 4.3.5 we already explained how our tool LinEx is able to extract information by querying e.g. the */proc* filesystem. It is not only possible to retrieve this information, but also to set new values to modify the behavior. For example, the file *nf_conntrack_udp_timeout* holds the UDP timeout value that can be set to an arbitrary new value by issuing the command: *echo "newValue" >nf_conntrack_udp_timeout*.

To describe a complete instance of a middlebox and its desired behavior we use our information model. It covers all relevant parameters and an instance of the model can be created by using an XML description as shown in listing 4.1. To also deploy a given topology we implemented a virtualized testbed as depicted in figure 4.16. The privileged domain 0 is used for managing virtual machines and networks. Here, three

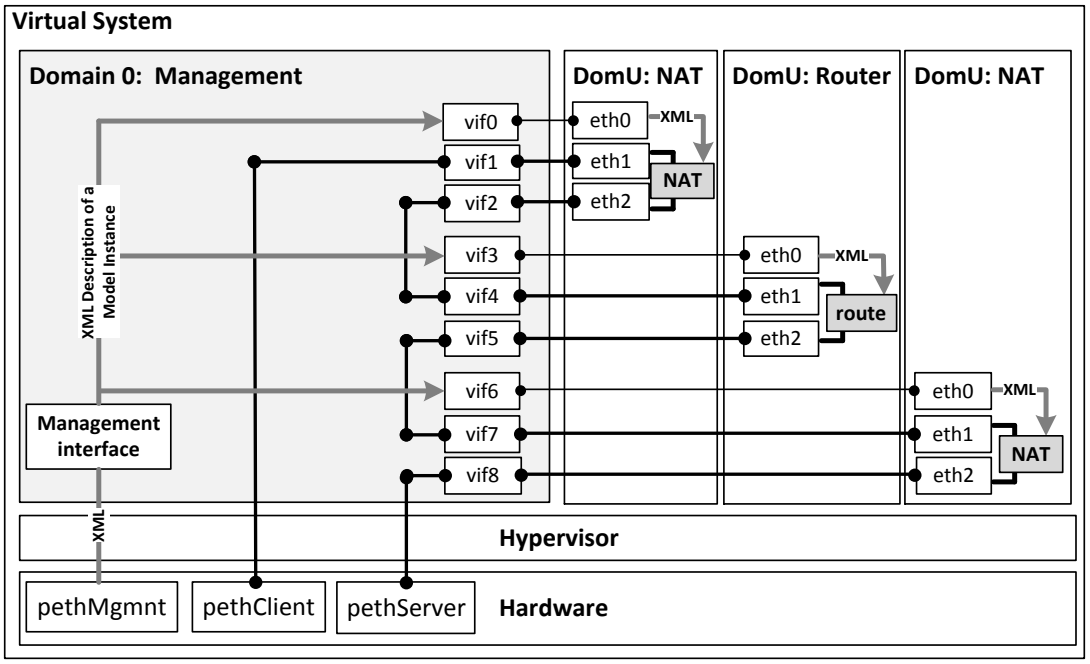

Fig. 4.16: Architecture of our virtualized topology generator

physical interfaces are used: *pethMgmnt* for management purposes (such as providing the configuration file), *pethClient* to connect the test client and *pethServer* to connect the test server. The test client and server could also be run inside a virtual instance, but for the sake of simplicity, our example assigns physical interfaces to them.

A sample configuration skeleton that is used to describe a middlebox topology is shown in listing 4.2. The machine configuration may come from a pre-defined template, but we choose to directly pass the XEN[5] configuration file to it. This configuration is then taken by the management interface and passed to XEN in order to create the new machine. The middlebox specific part following our information model is passed to the virtual machine image, which runs a parser that understands our model. The individual parts are extracted and the desired behavior is set up. Table 4.2 gives an overview of how it is possible to transfer an instance of the middlebox model into an actual Linux configuration.

```
<MachineDescription>
        <VM name="NAT1">
                ... (XEN machine configuration)
        </VM>
</MachineDescription>
<MiddleboxDescription>
        <n:Middlebox xmlns:n="http://example.org/MiddleboxSchema" ...>
                ... (description following our information model)
        <n:Middlebox/>
</MiddleboxDescription>
<VirtualNetworkDescription>
        ... (Bridging between virtual and real network)
</VirtualNetworkDescription>
```

List. 4.2: Configuration skeleton for the topology generator

[5] http://www.xen.org

Information Model	Linux Configuration
NetworkInterface Element	
ipv4:IPAddress	ifconfig command file: /etc/network/interfaces
ipv6:PrivacyExtension	add $net.ipv6.conf.eth0.use\_tempaddr = 2$ to /etc/sysctl.conf
ProtocolLayers Element	
Protocol:IP	route command
Stateful Element	
StateTable:TableSize	/proc/sys/net/nf_conntrack/nf_conntrack_max
StateTimer:UDP	nf_conntrack_udp_timeout
StateTimer:UDPStream	nf_conntrack_udp_timeout_stream
StateTimer:TCP-Established	nf_conntrack_tcp_timeout_established
Filtering Element	
State-based:Independent	iptables -t nat -A PREROUTING -j DNAT ...
Policy-based	iptables myPolicy -j Drop or Accept
Protocol-based	iptables myProtocol -j Drop or Accept
Translation_and_Modification Element	
Binding:NATBinding:EnpIndep.	iptables -t nat -A POSTROUTING -j SNAT ...
Binding:NATBinding:ConDep.	iptables -A POSTROUTING -t nat ... –random

Tab. 4.2: Examples for transfering our model instance to an actual configuration

4.5 Field Test

With the algorithms designed above (section 4.3) and the ability to set up pre-defined middleboxes and topologies in order to evaluate them (section 4.4), we now have all components for the analysis of middleboxes in the Internet. To cover as many different middleboxes as possible, we chose to conduct a public field test where we ask volunteers to participate and run the tests on their computers evaluating their middleboxes. The gathered results not only help us to understand middlebox behavior, but also to design algorithms for coping with them. This section presents the field test in detail. We first cover related work in section 4.5.1. Section 4.5.2 then presents our requirements and lists our contributions. Section 4.5.3 focuses on selected details of the implementation and the deployment. Finally, the results are presented in 4.6.

4.5.1 Related Work

Similar field tests covering network and middlebox behavior have been done in the past. In [48] a UDP hole punching algorithm is evaluated using a small number of devices in the wild. TCP middlebox behavior is tested in [60]. TCP traversal is harder than UDP and its success is dependent on the behavior of the middlebox. The authors therefore look at different behavioral issues, such as error handling and unusual packet sequences. Based on these results they present an algorithm that works for them with 89% of their tested middleboxes. [68] looks at home gateway characteristics such as TCP/UDP timeouts, DNS handling and ICMP messages. [76] tests basic UDP behavior like STUN classifications [133], port mapping and filtering.

The German Federal Office for Information Security (BSI) conducted a study covering 36 routers about the support of DNSSec in home routers [35] and found out that only 25% of their tested devices fully support DNSSEC [161]. [30] implements a *NAT-check-functionality* method in their existing P2P network and studies UDP characteristics of NATs and firewalls within their network. The tests covers UDP timeouts, as well as STUN results of 3500 unique peers. Within their P2P network, almost 90% of the peers are behind a stateful middlebox. But all these studies have one thing in common, that they only test a very small number of devices or the conducted tests and algorithms only cover a few issues. A test with a large result set covering behavior, traversal, as well as the detection of cascaded middleboxes has never been conducted.

Netalyzr [87] is an example of a large scale field test, which was designed as a useful tool for analyzing the current network connection, e.g. in case of a network problem. The web-based test covers many networking aspects such as latency, bandwidth, HTTP proxying, caching and also a basic NAT detection test. In their paper they refer to 130,000 measurements from 99,000 public IP addresses. The goal of the HomeNetProfiler [33] is to gather networking information of home routers to eventually improve the usability of home networks. The authors use the UPnP protocol for querying the home router and to collect middlebox related data such as the external IP address, access link capacity, buffer sizes, as well as the packet loss. In combination with Netalyzr the authors collected a large number of data from 120,000 homes and found many problems related to broken UPnP implementations. NetPiculet [158] was designed to test firewall and NAT policies, such as timeouts, filtering and mappings in mobile networks. This was realized using a smartphone application and the test was run in a large number of cellular provider networks. As a result the authors discuss the impact of their findings for mobile devices, e.g. battery draining due to a denial of service attack.

4.5.2 Requirements and Contributions

The usability of the field test for potential volunteers is essential for getting a large number of test results. A perfect solution would run on all major platforms without requiring the user to install any additional software. However, a trade-off between usability and functionality has to be found. Many algorithms and tests call for superuser privileges, which might prevent many users from running it. Additionally, an instant feedback and benefit must be provided to the user to draw their attention. Table 4.3 presents our requirements and shows the contributions that help satisfy them. The following section 4.6 then presents the results and findings of our measurements in detail.

	Requirement	Contribution
User Requirements		
R1	Usability	Web-based Approach
The test must be easy to use, should not require additional software, and it should be platform independent. Thus, a web-based approach was chosen.		
R2	Instant Feedback	Pretesting and Parallelization
The test should give an instant feedback and inform the user about his test result. We run all time critical tests in parallel and introduce optimization techniques for the topology test.		
R3	User Privileges	Logic on Server Side
To avoid the need for RAW sockets and root privileges, the logic is completely implemented on the server side.		
R4	Security and Privacy	Encrypted Storage
To prevent the stealing of sensible data, all results are encrypted on the server side and results are only shown in an aggregated way.		
Test Provider Requirements		
R5	Scalability	Distributed Backend
The server side must be able to handle a large amount of clients. Multiple instances and DNS load-balancing help to distribute the load and guarantee path-stability.		
R6	Update Management	Web-based Approach
New tests and updates should be easily integrateable.		

Tab. 4.3: Requirements and contributions for our field test

4.5.3 Design and Implementation

The website *http://nattest.net.in.tum.de* hosts our software, provides information about the conducted tests and presents an overview of the results in an anonymized way. During the last years, four different versions of the test software were implemented (as described in the next section, only two resultsets are considered in this thesis). All implementations follow the approach as depicted in figure 4.17: Visitors of the website download the software, run the tests with one of our test servers, report their results back to a database server and receive an aggregated result of their testrun in return. *Calculate result dependent on the logic* means that either the client or the server implements the logic for deciding about the results of the test. For example, the result of the hole punching test can only be determined by the client, while the result of the topology test is calculated by the server. If results are gathered on the client side they have to be committed to the test server as the final step of a testrun.

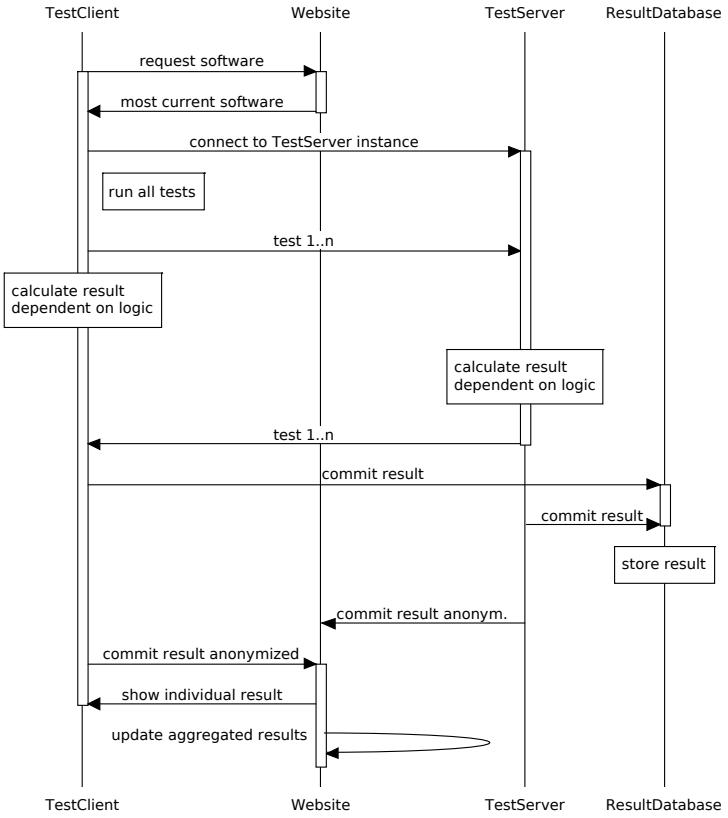

Fig. 4.17: Sequence diagram of our field test software

The first instance of a field test was developed as part of the authors diploma thesis [99] and was used to evaluate 79 NAT devices in Germany. The implementation was based on Linux and had to be compiled after downloading by potential participants before running it. It did not need superuser privileges. This approach was chosen due to the limited time available and due to the relatively easy implementation. However, many potential users could not be reached. In order to get more results, a Windows version was developed afterwards and first results were published in [103] and [104]. The approach of maintaining two individual implementations was not feasible and led to a complete redesign of the test. The results of these first implementations are not considered in this thesis.

A natural evolution of these clients was based on libpcap[6] and designed in a way to run on Windows and Linux. This was achieved by only using a text-based user interface and by putting platform dependent code into *ifdef* directives. For the Windows version a precompiled binary file together with libpcap was put into one zip-file that had to be downloaded. The libpcap-based test required superuser privileges to run tests that are based on RAW sockets, such as those for testing protocol-based filtering behavior. This was a major hindrance for many potential participants since they did not only have to grant administrator rights to the test software, but also disable the internal Windows firewall since it was blocking outgoing RAW packets. Parts of the results of the libpcap-based test were published in [100] and [105]. With the introduction of Windows 7, many tests failed due to the increased default security level and we had to redesign our tests again.

[6] http://sourceforge.net/projects/libpcap/

Web-based Implementation

A screenshot of the most current (and final) version of our field test is shown in figure 4.18. The tool *NATAnalyzer* is part of the measurement initiative *measr.net* of the Chair for Network Architectures and Services at the Technical University of Munich, Germany.

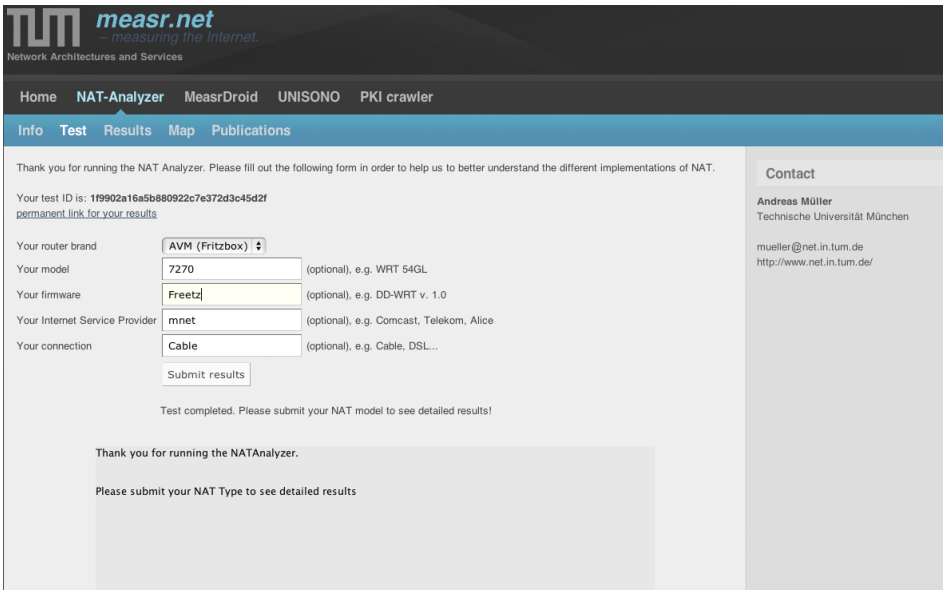

Fig. 4.18: Screenshot of the web-based NATAnalyzer

The test suite is implemented as a JAVA applet and runs on many common platforms. As the most important ones we have successfully tested our software with the operating systems Windows XP/Vista/7, different versions of MacOS 10 and Ubuntu/Debian Linux with all common browsers such as Google Chrome, Firefox, Internet Explorer, Safari and Opera. The frontend is implemented in HTML5 and makes an extensive use of Javascript and especially the *jquery*[7] library for asynchronous AJAX requests. The backend is implemented in *Python*[8] and uses the library Scapy[9] for sniffing and manipulating packets. Due to the restriction of JAVA applets that neither allow sending RAW packets nor setting the TTL of IP packets, all tests had to be designed to require as little resources and permissions on the client side as possible. Therefore, for certain tests we had to implement more logic and complexity as necessary when assuming that superuser permissions are available. For example, our hole punching algorithms require to set the TTL field of the IP protocol, but since this is only possible as a superuser, we designed the tests in a way that the server simulates the expiration of certain packets, e.g. by dropping packets or by answering packets with pre-defined packets from a second IP address (e.g. drop initial UDP packet and answer with ICMP TTL exceeded from a different IP address).

Figure 4.19 shows the steps of a single testrun using the web-based implementation. First, the participant visits the website, and executes the JAVA applet (the test client) in the browser (step 1+2). The JAVA applet then chooses a nearby test server based on the current IP address (test servers are currently located in Munich, Germany and

[7] http://jquery.com/
[8] http://www.python.org/
[9] http://www.secdev.org/projects/scapy/

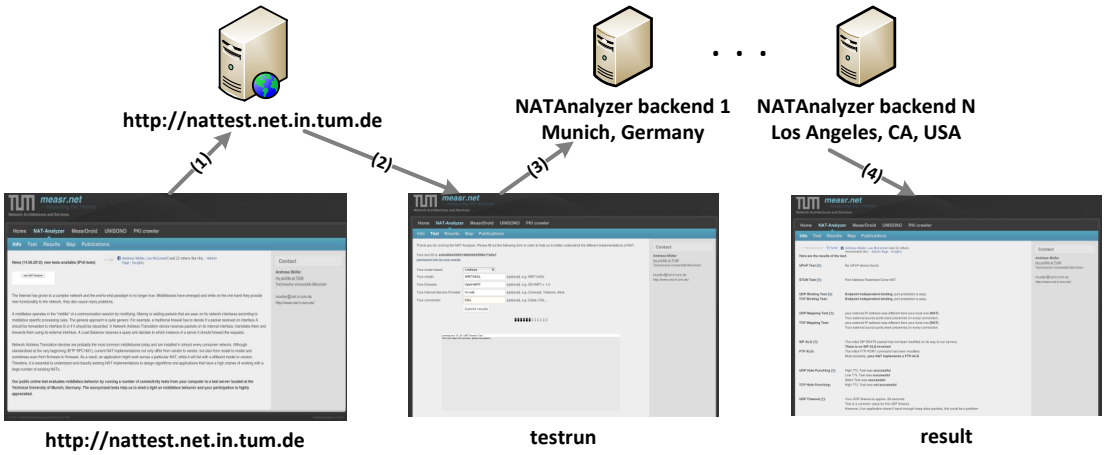

Fig. 4.19: Web-based approach

Los Angeles, CA, USA) and runs the tests (step 3). During a testrun we ask the participant to provide information about the home router that is tested, about the Internet Service Provider and about the connection to the Internet. Although these fields are totally optional, many participants provide at least parts of the information (see next section). Once the test is done the client commits the results to our database and an aggregated result is immediately shown to the participant. These results also contain useful information about possibilities to tweak the router settings if some results are an indicator for poor performance with some popular applications. For example, if the timeout of the middlebox is shorter than the average, we suggest to send more keep-alive packets and provide settings for common VoIP/SIP applications. The results are committed to a MYSQL database that is not directly accessible from the Internet.

Android-based Implementation

Parts of the NATAnalyzer software are currently being integrated into the *MeasrDroid* Android framework developed at the Chair for Network Architectures and Services at the Technical University of Munich. The goal of MeasrDroid is to utilize Android smartphones to collect location-based information about network related parameters. The individual parameters can be divided into passive measurements (e.g. getting the location from the Android phone) and active measurements, such as round trip times and eventually middlebox behavior.

While NATAnalyzer mainly measures fixed line connections, MeasrDroid targets mobile networks and middleboxes located at the ISP. By default, MeasrDroid executes its manyfold tests every 15 minutes and therefore forbids to run time consuming algorithms. Thus, we integrated our binding and mapping behavior tests to gather information about port allocation algorithms of provider-based LSNs and to verify findings of [158]. Additionally, the topology analysis algorithms were integrated. Since the algorithms were already available as JAVA code, porting it to a MeasrDroid module was achieved with only minor changes. The public release of the MeasrDroid framework (via the Google Play Store[10]) is planned for 2013 and its results will help to shed more light on the behavior and impact of Large Scale NATs in mobile environments. In this thesis we only mention preliminary results if applicable.

[10] https://play.google.com

4.6 Results and Discussion

The following sections present the results of the conducted field test. We start in section 4.6.1 by describing our two testsets. We present general findings such as the distribution of the test results and explain biases that are inevitable when requiring user participation. Section 4.6.2 then starts by classifying our testsets using the STUN algorithm. Afterwards, we compare the results to our binding measurements and give an estimation to which extent today's middlebox traversal algorithms are supposed to function. When looking at connection dependent binding strategies, we identified clear patterns that can be used to develop new algorithms in order to also traverse many more middleboxes compared to the state of the art. Section 4.6.3 then analyzes mapping behavior and presents new categories for describing the *Translation_and_Modification:Mapping* element in detail. Additional behavior results, such as the ICMP, timeout and mapping table analysis are presented in section 4.6.4. Section 4.6.5 presents success rates for different behavior-based middlebox traversal mechanisms. The results support our arguments that parameterizing traversal algorithms according to the behavior of the participated middleboxes help to significantly increase their success rates. UPnP, SCTP and ALG results are presented in section 4.6.6. After presenting our topology results in section 4.6.7, we conclude with the lessons learned in section 4.6.8.

4.6.1 Testset Description

The conducted field test consists of many individual experiments as listed in table 4.1. Some of them have been defined at the beginning of our research, others were added later on or were modified and adapted to the most current findings. Additionally, the individual implementations of the field tests (see above) produced disjoint result sets. Figure 4.20 shows which of our algorithms were considered for which testset. As mentioned above, our results focus on the web-based and libcap-based implementation and preliminary results of the Android-based implementation are only mentioned if applicable.

Table 4.4 shows the number of individual tests for each testset. Testset 1 (libpcap-based) contains 2,651 valid entries from 2,232 unique IP addresses, whereas testset 2 (web-based) contains 1,717 valid tests from 1,690 unique IP addresses. The reason for having multiple tests for the same IP address is not only due to dynamic IP addresses that are reassigned to different customers after a specific amount of time (this did happen more often than expected), but also because many volunteers tested the same box with different parameters within a short period of time. We observed two different user behaviors: The first group runs the test multiple times without changing anything in order to verify their results. The second group runs the test once and changes some parameters for additional runs. For example, after the first test a participant may enable UPnP or set up a static port-forwarding entry to validate his settings. Surprisingly, many subsequent tests from the same IP address show a different behavior, because the participant changed the firmware of the model (e.g. Tomato vs. OpenWRT) to find out which one behaves "better". In one particular case we identified four results all coming from the same public IP address (assigned to a business in Hongkong) and same local interface within a short period of time (five hours). The results however show four different models (three found by UPnP and one that was specified manually). Obvious duplicates (same behavior, e.g. UPnP model, internal IP address and local network interface from the same IP address within a short period of time) were manually removed from testset 2 in order to not distort our results. However, we also double-checked our

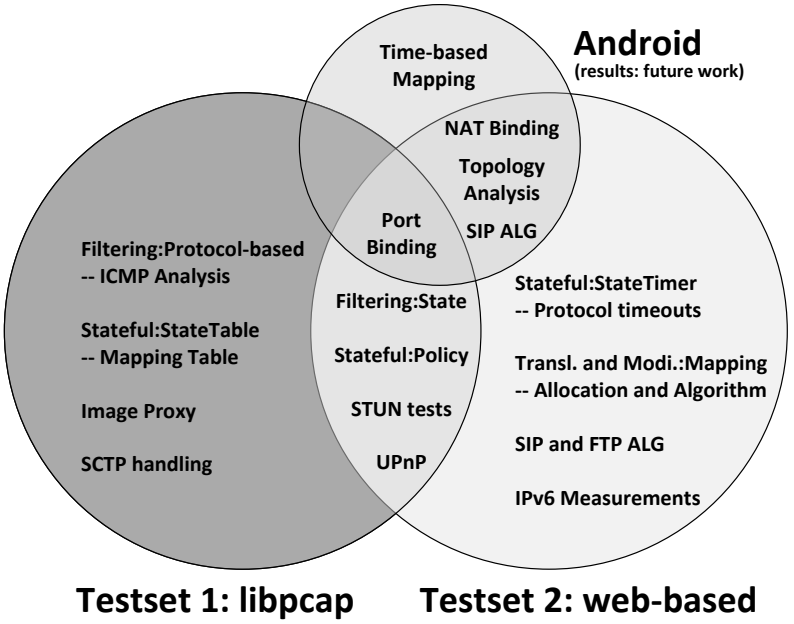

Fig. 4.20: Our three testsets and their individual tests

findings by querying the whole testset and the percentages differed marginally, while all finding remain the same. In addition to the fact that the number of middlebox vendors is limited, we are convinced that our result set (although it contains biases as described below) is large and varied enough to actually make a rather general statement about the behavior of middleboxes, especially the ones that are located on the edge of the Internet.

Identified Biases

The main problem of a field test relying on the participation of volunteers is that it is not possible to influence the test group in a way to get a representative result set. Thus, all of our testsets contain certain biases. The most obvious one is referred to as a "geek-bias" (following the terminology of [87]) and since our test targets rather technical users this is inevitable. For example, compared to common published marked shares[11] we found the number of Linux and Apple users to be much higher: Linux: 3.24% in our test vs. 1% market share and Apple: 16.05% in our test vs. 7%. Additionally, 48.35% of all participants of testset 1 and 2 specified their firmware and even 66.07% specified the model of their home router. An average user may be able to identify the model, but in order to find the exact firmware it is necessary to log into the website of the router and extract the information, something an inexperienced average user is not capable of. Finally, almost 15% of the 290 latests measurements of testset 2 were able to establish an IPv6 connection.

Another identified bias is based on the location of the participants. For testset 1 26.71% of all data sets are from Germany and 16.94% from the US. For testset 2 the top 3 countries are the same with a distribution of 23.13% for Germany, 20.65% for the US and 7.87% for the UK. A typical peak that also created a bias can be seen a few hours and days after asking students during a related lecture at the Technical

[11] http://www.netmarketshare.com/

Testset 1 (libpcap)		**Testset 2 (web-based)**	
Valid Tests			
2,651		1,717	
Unique IP Addresses			
2,323		1,690	
ISPs			
624 (> 10 sessions: 51)		522 (32)	
Top 8 ISPs			
Deutsche Telekom	9.8%	Deutsche Telekom	7.03%
Chunghwa Telecom	3.3%	Comcast Cable	3.81%
Comcast Cable	2.8%	Arcor AG	2.86%
Arcor AG	2.7%	Virgin Media	2.78%
Alice DSL	2.2%	Kabel Deutschland	2.68%
Virgin Media	1.8%	M-net GmbH	2.2%
Road Runner	1.5%	Cox Communic.	1.51%
freenet	1.5%	British Telecom	1.49%
Countries			
89 (> 10 sessions: 26)		77 (29)	
Top 5 countries			
Germany	26.7%	Germany	23.13%
United States	16.9%	United States	20.65%
United Kingdom	5.5%	United Kingdom	7.87%
Taiwan	3.7%	Netherlands	3.96%
Canada	3.6%	Korea	2.9%
Top 5 Middlebox Vendors as provided by the users, UPnP findings in brackets			
Linksys	14.59%	Netgear	8.9% (14.89%)
D-Link	10.98%	Linksys	8.9% (9.43%)
Netgear	8.34%	D-Link	6.71% (10.66%)
AVM Fritzbox	6.79%	AVM Fritzbox	6.62% (17.2%)
T-Com	4.75	Zyxel	3.87% (2.67%)

Tab. 4.4: Testsets for the libpcap-based (1) and the web-based implementation (2)

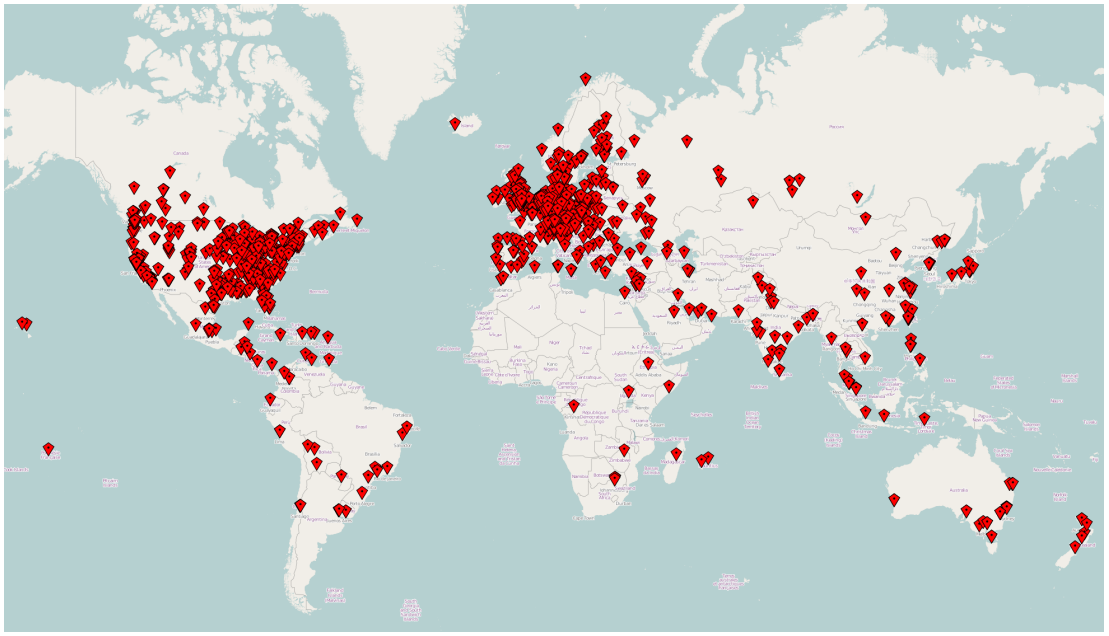

Fig. 4.21: Geographical distribution of testset 1 and 2[13]

University of Munich to run the test. In fact, according to the IPInfoDB[12] database, 10.05% of testset 1 and 4.02% of testset 2 is coming from Munich, Germany. Additional announcements were made in some english speaking portals, which explains the high number of users from the US, the UK and Canada. The geographical distribution is depicted in figure 4.21.

4.6.2 Binding and Filtering Behavior Results

Translating middleboxes implement a binding strategy that describes which external mappings are allocated for an internal one. More precisely, we are interested in NAT Binding and Port Binding behavior that is part of the *Translation_and_Modification* element. We first start with categorizing our results using the STUN protocol, since this is a common way of classifying stateful translating middleboxes and it reveals state-based filtering behavior as described in the *Filtering* element of our information model. Afterwards, we look at the results of our algorithms and draw conclusions regarding the applicability of our results for the traversal of middleboxes.

Classification according to STUN

The first version of the STUN protocol as defined in [133] introduced four different implementations that can be used to classify middlebox behavior: Full Cone, Address Restricted Cone, Port Address Restricted Cone and Symmetric (see section 3.3.3 for more details). For both testsets we ran the STUN algorithm (used libraries: VovidiaStun[14] for testset 1 and JSTUN[15] for testset 2) and extracted the STUN types for all test entries. The results are depicted in figure 4.22. A Port Address Restricted Cone NAT (PAR) was discovered in 72% of all results of testset 1 and 57% of testset 2,

[12] http://ipinfodb.com/
[13] Figure: ©OpenStreetMap contributors: http://www.openstreetmap.org/copyright
[14] http://sourceforge.net/projects/stun/
[15] http://jstun.javawi.de/

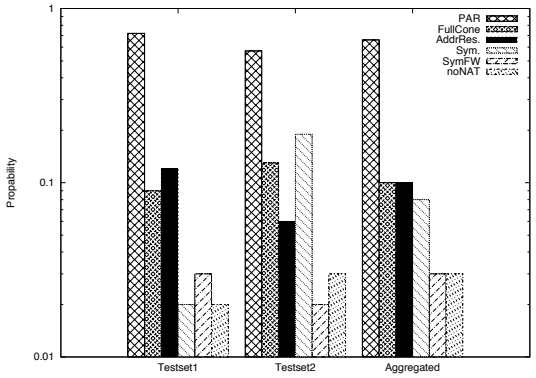

Fig. 4.22: STUN classification

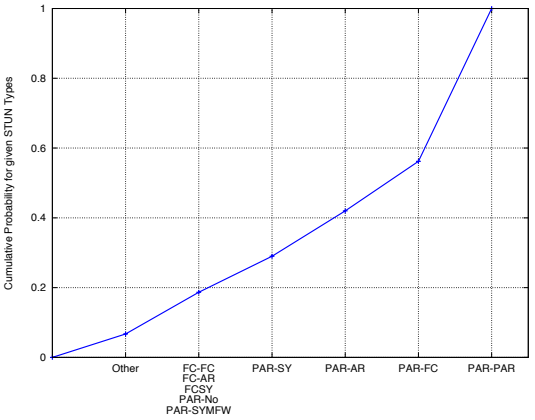

Fig. 4.23: STUN constellation

leading to an aggregated result of 66%, which is two thirds of all tested middleboxes. Full Cone NATs were found in 10% (aggregated) and Address Restricted Cones also in 10%. Only 3% of our tests do not implement address translation and provide an open access to the Internet. The number for Symmetric NATs differs dramatically between testset 1 (2%) and 2 (19%), leading to an aggregate of 8%. The reason for this large difference is hard to assess. Of course it could be due to a different testset, but since most general conditions such as the distribution (number of ISPs, number of countries) and size are similar, it could also be due to the different STUN libraries. However, besides Symmetric NATs and Port Address Restricted Cones the other results only differ slightly. It seems like some of the Port Address Restricted NATs were classified as Symmetric ones for testset 2 and vice versa. Since the difference between the two types is the binding strategy, the issue will be further investigated when looking at the binding behavior in more detail.

Today, the success rate for establishing a session between two peers both behind a filtering and translating middlebox mainly depends on the behavior of the service's middlebox. If the service is not able to predict external ports, a connection will most likely fail. The rather high number of Symmetric NATs indicates that a connection will fail in approx. 8% (19% for our second testset). We argue that for many applications it would be possible for the "client" and "server" to switch roles and to establish the session in the other direction. Thus, the probability derived from the aggregate of testset 1 and 2 for a certain constellation for two arbitrary peers in the Internet is depicted in figure 4.23. In almost 90% of all constellations a Port Address Restricted Cone NAT is involved. In 43.8% both peers implement this behavior. In less than 1% both endpoints implement a symmetric behavior. This means that many applications, if designed properly, would be able to communicate across middleboxes utilizing legacy traversal mechanisms such as hole punching in a more parametrized and more structured manner. This is our goal for chapter 6.

NAT and Port Binding

Port Binding for testset 1 was measured using the STUN libraries internal functionality for comparing external ports. As a result, 79.89% of all middleboxes preserve their ports. This rather high number is confirmed by testset 2, where 71.45% for UDP and 70.28% for TCP implement port preservation.

For testset 2, we then extracted the percentages for the STUN categories (see table

4.5) and observed some interesting findings: While Restricted Cone NATs (AR and PAR) preserve their ports for outgoing UDP as well as TCP packets in more than 86%, the number for Full Cone NATs (53.57% for UDP and TCP) and Symmetric NATs (17.48% for UDP and 17.83% for TCP) are much lower. The numbers for Symmetric NATs are as expected, but the reason for the Full Cone NAT behavior is unclear. A possible explanation could be that due to the fact that Full Cone NATs implement a rather unrestrictive filtering behavior (all packets are forwarded independent of their source addresses), vendors might want to introduce some obfuscation by not preserving port numbers. Additionally, since the state table of Full Cone NATs does not contain the destination IP address and port (independent filtering), port preservation can only take place if the port is not in use by another client. While NATs implementing a restricted filtering strategy are able to multiplex based on a combination of the source port and the destination (and are therefore able to allocate the same source port to different connections), for Full Cone NATs the external source port is the only unique identifier for incoming packets.

	All	Full Cone	AR	PAR	Symmetric
Port Binding					
Port Preservation	UDP: 71.45%	53.57%	86.9%	88.25%	17.48%
	TCP: 70.28%	53.57%	86.9%	87.79%	17.83%
No Preservation	UDP: 28.55%	46.43%	13.1%	11.75%	82.52%
	TCP: 29.72%	46.43%	13.1%	12.21%	82.17%
NAT Binding					
Endpoint Indep.	UDP: 78.45%	78.06%	94.05%	99.77%	3.85%
	TCP: 75.25%	65.82%	89.29%	96.77%	9.79%
	both: 72.34%				
Conn. Dependent	UDP: 21.55%	21.94%	5.95%	0.23%	96.15%
	TCP: 24.75%	34.18%	10.71%	3.23%	90.21%
	both: 27.66%				

Tab. 4.5: Detailed binding behavior results of testset 2

We then measured the NAT binding behavior of testset 2. According to our information model, *Translation_and_Modification:Binding:NATBinding* may have two values: Connection Dependent and Endpoint Independent. As shown in table 4.5 independent binding was observed in 78.45% for UDP and 75.25% for TCP, which indicates that our observed numbers for Symmetric NATs are correct. If the middlebox implements a connection dependent binding for UDP it almost always also implements a connection dependent binding for TCP. Only 9.2% of all tested middleboxes implement a different binding behavior for TCP and UDP. The results show that port prediction using the STUN algorithm works with more than three quarters of all middleboxes without any additional effort.

However, in order to verify our results we extracted the binding behavior for all four middlebox categories. Since the categories are defined by their binding (and filtering) behavior, we didn't expect any surprises. For Symmetric NATs the connection dependent rate was expected to be 100%, while the other categories were supposed to

implement an independent binding strategy. However, the results show that this is only true for Restricted Cone NATs (94.05% for AR and 99.77% for PAR). For Full Cone NATs 21.94% use a connection dependent binding and 3.85% of the Symmetric NATs (according to the STUN results) implement an independent binding strategy. When looking at the detailed results we didn't find any clusters, specific models or connections. A reason for the difference between the STUN result and the binding test could be cross-traffic, which we are neither able to identify nor to influence.

Although the STUN classification was only done for UDP, we are still interested in the results for TCP. The percentages are similar than those for UDP except for Symmetric NATs, where a much higher number implement an endpoint independent binding for TCP (3.85% vs. 9.79%). Altogether, 1.94% of all middleboxes that were identified as Symmetric NATs implement an independent binding strategy for UDP and for TCP. These findings encourage us to mainly rely on our endpoint independency tests instead of the STUN results and affirm the general assumption that the STUN classification does not cover all middlebox categories: it was removed from the most current STUN RFC with the following explanation: "Classic STUN's algorithm [...] was found to be faulty, as many NATs did not fit cleanly into the types defined there." [130].

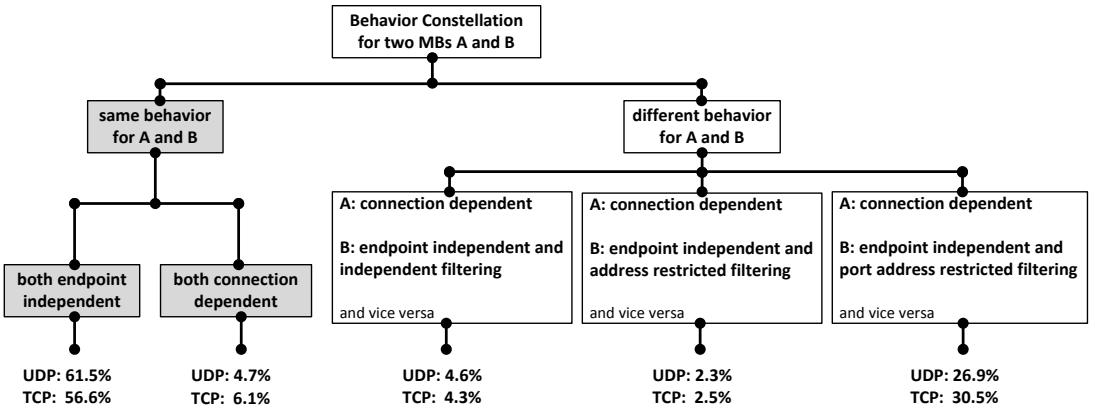

Fig. 4.24: Behavior probability for two arbitrary hosts of our testset 2

With this knowledge we are able to calculate correct probabilities for constellations of two arbitrary hosts of testset 2 that both implement endpoint independent binding, which is a prerequisite for state of the art behavior-based middlebox traversal. The results as depicted in figure 4.24 show the reason for today's rather poor success rate for middlebox traversal between two arbitrary hosts, both behind NAT, in the Internet: In the state of the art only 61.5% (56.6% for TCP) of the constellations provide the essential prerequisites that allow behavior-based traversal.

We then calculated the numbers for the constellations where only one host implements a connection dependent binding strategy. For the other host implementing an endpoint independent strategy we also focused on the filtering behavior. The results were calculated using our binding results (instead of STUN's results) for determining endpoint independent and connection dependent binding types and the STUN algorithm for the filtering results (only those that implement an independent binding strategy according to our test). For port address restricted filtering STUN types PAR, SYM and SYMFW were considered. We argue that if the constellation and the behavior is known, some of the constellations can even be traversed without requiring port predic-

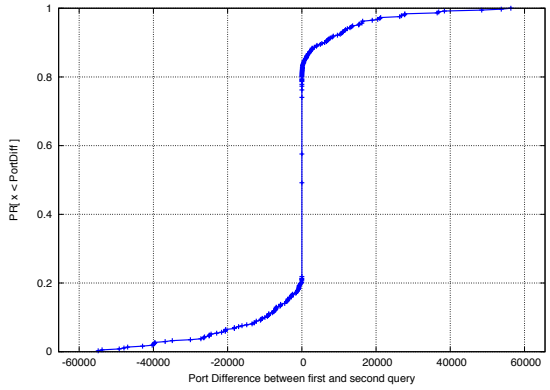

Fig. 4.25: UDP NAT binding

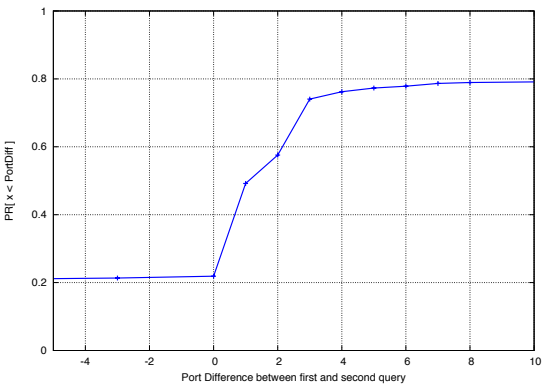

Fig. 4.26: UDP NAT binding details

tion, dependent on the filtering behavior. All we have to make sure is that the host that does the hole punching (the "service") is the one that implements endpoint independent binding. If not, we propose to "swap roles" and turn around the connection establishment. For example, a connection dependent binding strategy on the requester side and an endpoint independent together with an independent filtering strategy on the service side means that once the hole is created all packets will be forwarded. The requester is therefore not required to predict any external port. In case of address restricted filtering, the requester only has to predict its external IP address, but not its port since the service only filters based on the IP address. For port address restricted filtering, this approach does not work. However, if it would be possible to predict a port range instead of a concrete port, the effort for creating a connection would decrease dramatically.

Finding 1: *Today, only 61.5% (56.6% for TCP) of all middlebox constellations implement an endpoint dependent binding strategy, a prerequisite for state of the art behavior-based traversal. When also considering filtering behavior and by choosing roles carefully, it is possible to increase this number to 68.4% for UDP and 63.4% for TCP.*

In order to further improve these numbers, we will now take a closer look at the connection dependent binding implementations with the goal of identifying binding patterns. When comparing external bindings that were returned by our test server (for possible results see section 4.3.1), in less than 1% (of the connection dependent middleboxes) not only the external port, but also the IP address differs, thus implementing a non-pooling address behavior. For one such result coming from an IP address of Global Village Telecom serving a customer in Sao Paulo, Brazil, we were able to query the external IP address of the first hop middlebox via UPnP, which turned out to be private. This is a strong indicator that the mentioned provider deploys LSN in its network.

We then calculated the differences of the allocated external ports for all connection dependent mappings. For example, if the first connection returns the mapping *131.159.14.1:20000* and the second connection returns *131.159.14.1:20001*, the difference is 1. If the second port is smaller than the first one, it counts as a negative difference. The results were grouped and figure 4.25 shows the CDF of the port differences for UDP and figure 4.27 for TCP. The left hand side shows that negative differences are observed for approximately 20% of our results and more than half of them are in the $-20000 : 0$ range. However, a significant cluster cannot be seen. A huge peak can be observed at around 0 and due to the scale it is hard to interpret. The more detailed

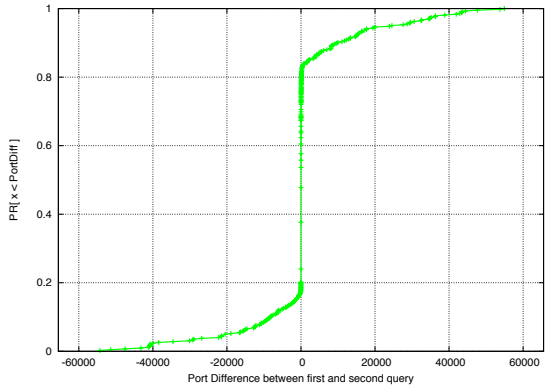

Fig. 4.27: TCP NAT binding

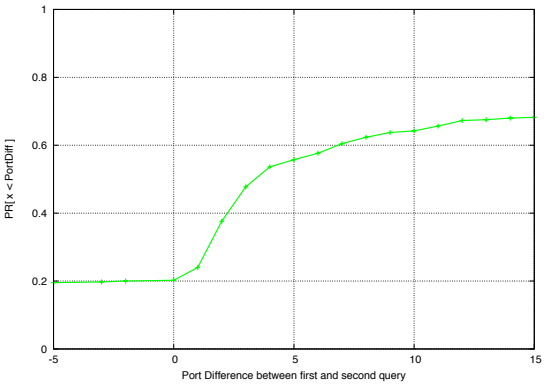

Fig. 4.28: TCP NAT binding details

figures 4.26 and 4.28 focus on a smaller range and reveal that a port difference of 1 can be observed with 27% of all connection dependent bindings. Other popular differences are 2 (8%), 3 (16.5%) and 4 (2.2%), covering 53.7% of all connection dependent binding middleboxes. A similar behavior is true for TCP, with popular differences of 2 (13.6%), 3 (10.1%), 4 (5.9%) and 5 (2.1%). A difference of 1 was only seen in 3.7%. The remaining question is which middleboxes will actually behave the same way for multiple queries and also when changing the source or destination addresses. More precisely, how stable is the port difference when changing parameters? Will the middlebox use the same port difference again and again or will it change it from time to time? Our results show that if the port difference is located in the $[0, 10]$ interval, the difference remains the same for all mappings independent on the source or destination. For other results the difference of the second mapping is not always related to the first one and therefore not predictable. However, if the middlebox implements port preservation, which is true for 20.54% (7.29% TCP) of all connection dependent bindings, it is not necessary to predict the external port because it won't change. When subtracting the number of middleboxes that already allow port prediction based on the binding strategy, an additional 2.7% (2.1% for TCP) of all connection dependent bindings preserve their mapping.

Finding 2: *Even if the middlebox implements a connection dependent binding strategy, in approx. 57% (44% for TCP) it is still possible to predict the external binding by analyzing binding patterns. When also considering connection dependent bindings that implement port preservation these numbers increase to 60% for UDP and 46% for TCP.*

4.6.3 Mapping Behavior Results

Port Mapping, also belonging to the *Translation_and_Modification* element, describes how external resources (transport layer ports in case of NAPT) are allocated to new mappings. When comparing the internal combination of an IP address and port with the external one we get four possible options as described in section 4.3.1: either the addresses and ports are different, only the address or the port is different or both parameters are the same. In case of a connection dependent binding strategy that does not allow querying an external service (e.g. STUN) for the external mapping, it may still be possible to predict mappings by analyzing port mapping strategies. For example, if a middlebox continuously increases the external mapping independent of the source address, n tests can be conducted to predict the $n + 1$ mapping.

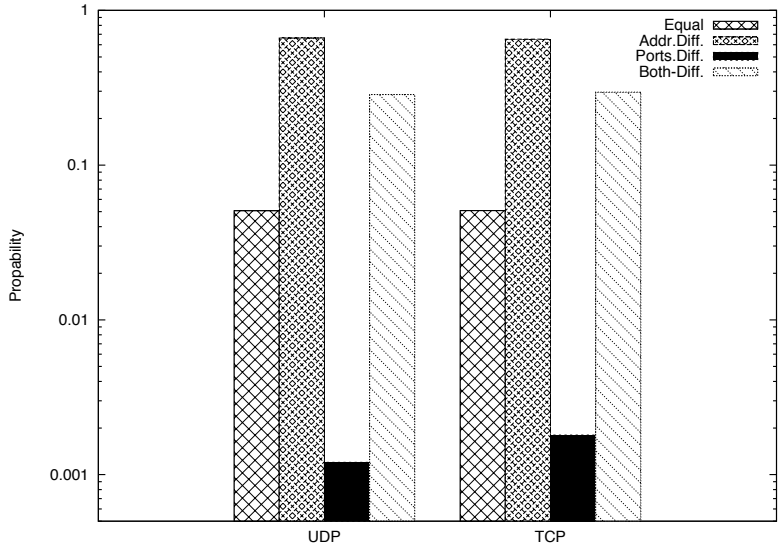

Fig. 4.29: Distribution of the mapping behavior

Figure 4.29 shows the distribution of the mapping behavior for UDP and TCP for testset 2. The reason for using a logarithmic y-axis is that the result set contains 2 middleboxes (0.0012% for UDP and 0.0018% for TCP), which actually use the same internal and external IP address, but different ports. This rather unusual mapping behavior is worth a closer look: we identified the first one as a result coming from a fixed-line provided by Portugal Telecom and implementing a Full Cone NAT according to STUN. However, according to our binding tests, a connection dependent binding with a port delta of three for UDP and four for TCP, independent on the source port, was found. Unfortunately, we could not determine the actual model or type of the middlebox, since neither UPnP nor the user himself did find or specify any information. The second result using different external ports but the same external IP address came from a DSL connection provided by GO[16] in Malta. STUN detected a Symmetric NAT and we also identified a connection dependent binding. The middlebox uses a fixed delta for allocating outgoing ports: three for UDP and four for TCP. Again, it was not possible to identify the model of the middlebox due to missing information that could have been provided by the participant.

In the next step we analyzed 33 individual mappings for each middlebox to understand the distribution in case the IP address and port number differ. We calculated the differences of all 33 external source ports (the number of gaps) and clustered them. For example, a middlebox that allocates the first starting port and continuously increases the external port by a fixed delta would carry a port mapping gap number of 1. If the number of gaps is 2 this could mean the following: either the continuos strategy is interrupted by one outlier (e.g. fixed delta of 1, but once a delta of 2), or the middlebox might implement a pattern using exactly two gaps (e.g. 3-4-3-4 or 3-3-4-3-3-4). Finally, a port mapping gap number of 2 may also mean that a middlebox uses a fixed delta dependent on the internal ports (e.g. 20,000(int):40,000(ext); 20,001:40,001; 30,000:50,000; 30,001:50,001; 40,000:60,000; 40,001:60,001). Figure 4.30 shows the cumulated probabilities for each port mapping gap number.

[16] http://www.go.com.mt/

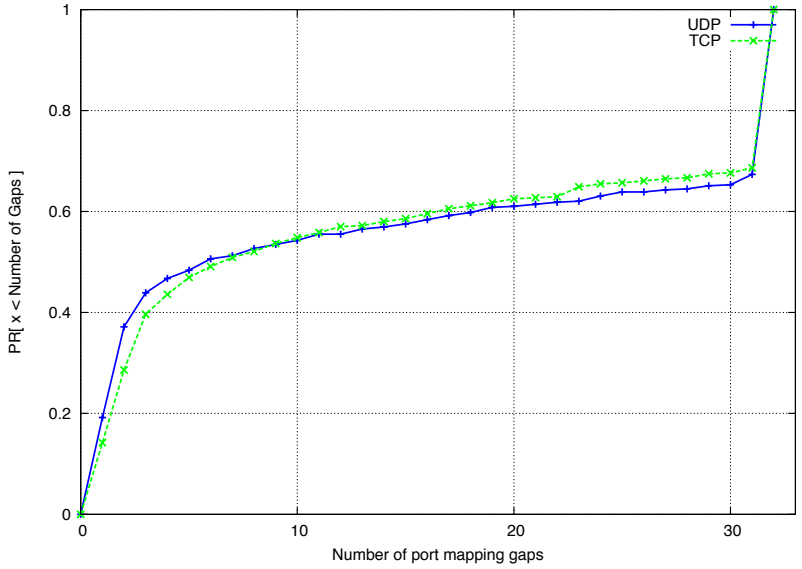

Fig. 4.30: CDF of the mapping behavior for middleboxes not preserving ports

Port Mapping Gap Number of 1: For UDP, 19.18% of all non-port preserving middleboxes utilize only one gap. This relatively large number of middleboxes continuously increases or decreases the external port by a fixed number independent of the internal source port. We call this category **fixed-delta independent** and show an example on the left hand side of figure 4.31 with a fixed delta of 3. When looking at the utilized deltas we only found three different ones: 88.3% (of the 19.18%) utilize a fixed delta of 3, 8.51% use a fixed delta of 1 and 3.19% use a fixed delta of 6. For TCP 14.2% only use one port mapping gap number. We again identified three different deltas: 95.8% use a fixed delta of 3, 2.8% a delta of 1 and finally 1.4% a delta of 8.

Port Mapping Gap Number of 2: 17.96% of all UDP non-port preserving middleboxes utilize a port mapping gap number of 2. When looking at the individual numbers we found that for 48.86% the reason for having a port mapping number of 2 (instead of 1) is one outlier. For example, instead of continuously increasing the external port by 1, it is increased by 2 for one arbitrary mapping. This outlier could be due to cross-traffic or be intended by the implementation, which is not possible to verify with our result set. However, this observation allows us to define our second category: **error-prone** mapping. 36.36% showed more than one outlier, but all of them were within the same port mapping gap number (e.g. always delta of 1, but twice a delta of 3). The number of outliers was still rather low, between 2 and 5, which still allows identifying a fixed delta. A port mapping gap number of 2 allows us to define further categories: 9.08% use the same delta as long as the internal port numbers are increased by the same delta. Once the internal port numbers jump (e.g. from 20,010 to 30,000), the external ones jump as well. Since our testrun implements two internal jumps, a port mapping gap number of 2 means that the external jumps carry the same delta. We call this third category **fixed-delta dependent**, since a fixed delta is dependent on the internal port numbers. The fourth category **pattern-based** was also discovered when analyzing a port mapping gap number of 2. In 5.7% a pattern such as (3, 3, 4) was identified.

Port Mapping Gap Number of 3: A port mapping gap number of 3 was discovered for 6.73% of our non-preserving middleboxes. For 45.45% a **fixed-delta dependent** was found, whereas the port mapping gaps on the edges (when the internal ports jump) differ. 26.7% jump to an unmotivated new value (e.g. by -90 for the first time and by 162 the second time), but for 73.3% the following two values were discovered: 9,990 and -22,778. The 9,990 is exactly the difference of the internal port and thus the middlebox also jumps by this value for the external one as above. However, since the port number field is only 16bits (65,536 port numbers) the middlebox has to wrap around and adjust the starting port number if the new port is greater than 65,536. Thus, whenever the middlebox reaches the end of the port range, it jumps back by 22,778 and continues with the initial algorithm. Most likely, the numbers above carrying a port mapping gap number of 2 also implement the same behavior. 48.48% carrying a port mapping gap number of 3 implement an **error-prone** mapping and for 6.07% a **pattern-based** strategy was found.

Port Mapping Gap Number of 4: For the 2.85% that carry a port mapping gap number of 4 all of them implement an **error-prone** strategy.

Port Mapping Gap Number of 5: 42.86% of the middleboxes implementing a port mapping gap number of 5 (1.63% of all non-port preserving results) implement a **pattern-based** mapping strategy. The pattern (1, -3 , 1, -3, 1, -3, 1, 13) JUMP (1, -3 , 1, -3, 1, -3, 1, 13) was observed twice and the middlebox was identified via UPnP as *D-Link DAP-1360*.

Port Mapping Gap Number > 5: For these port mapping gap numbers we could not identify any patterns. All of our results can be classified as **error-prone** and with an increasing port mapping gap number the mapping tends to randomize.

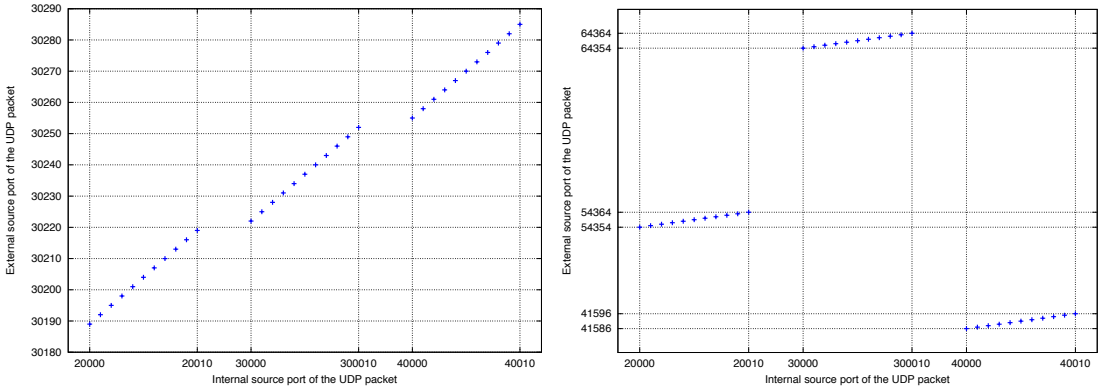

Fig. 4.31: Mapping schemas fixed-delta independent (l) and fixed-delta dependent (r)

By calculating and analyzing port mapping gap numbers we were able to identify three categories describing port mapping strategies: **fixed-delta**, **pattern-based** and **error-prone**. Each of the categories can be further divided into independent and dependent, describing the dependency of the internal port. Table 4.6 shows the categories together with the necessary parameters for describing them. The parameters can be used as an input for the *Translation_and_Modification:Mapping* element of our information model in order to precisely specify mapping behavior.

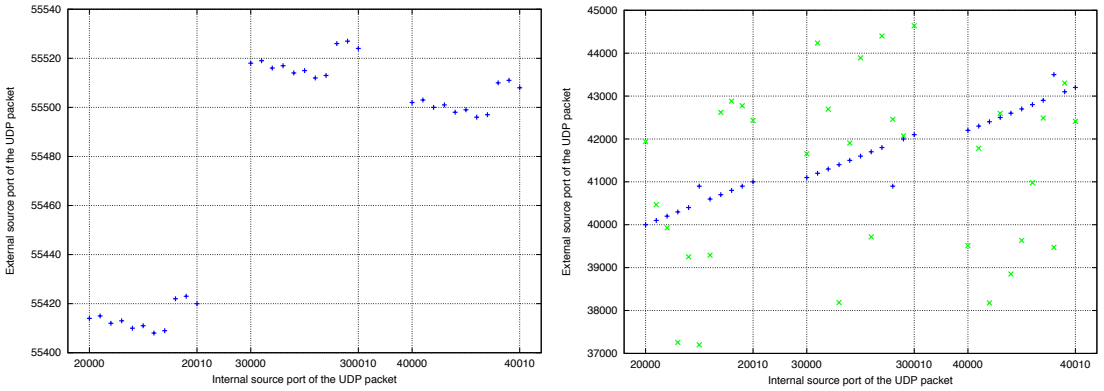

Fig. 4.32: Mapping schemas pattern-based dependent (l) and error-prone (r)

The first category, **fixed-delta**, is depicted in figure 4.31. An external starting port ($sPort$ according to the parameters in table 4.6) is allocated for the first internal port and is continuously increased by a fixed value Δ. On the left hand side of figure 4.31, Δ is only dependent on the time, but independent of the internal source port. This means, although there are two internal jumps from 20,010 to 30,000 and from 30,010 to 40,000 the external source port is still continuously growing by the same value. However, since port numbers are limited to 16bit a wrap-around has to be done once the external port approaches this limit. The wrap-around port can be specified using $wPort$. The right hand side of figure 4.31 depicts an example for a fixed-delta strategy that is dependent on the internal source port. Δ remains constant for all intervals, but as soon as the internal source port jumps, the external one jumps as well. We use a set of ports $\{sPorts\}$ for specifying the starting ports for each interval. Port prediction with fixed-delta mappings is rather straightforward. Δ can be discovered by a few consecutive test runs and once the initial sPort is known for the first test mapping, the successor can be calculated by $sPort + \Delta$.

Category		Description	Parameters
Fixed-Delta	I	continuos growth of Δ, ext. port is time dependent	Δ, sPort, wPort
	D	cont. growth of Δ, ext. port is dep. on internal one	Δ, $\{sPorts\}$, wPort
Pattern-based	I	iterative time dependent pattern	$\{\Delta_1, ..., \Delta_n\}$, sPort, wPort
	D	iterative pattern within a fixed interval	$\{\Delta_1, ..., \Delta_n\}$, $\{sPorts\}$, wPort
Error-Prone	I	fixed-delta with outliers or more or less random	Δ, sPort, σ, $[E_{min}, E_{max}]$, wPort
	D	within a fixed interval	Δ, $\{sPorts\}$, σ, $[E_{min}, E_{max}]$, wPort

Tab. 4.6: Identified mapping categories

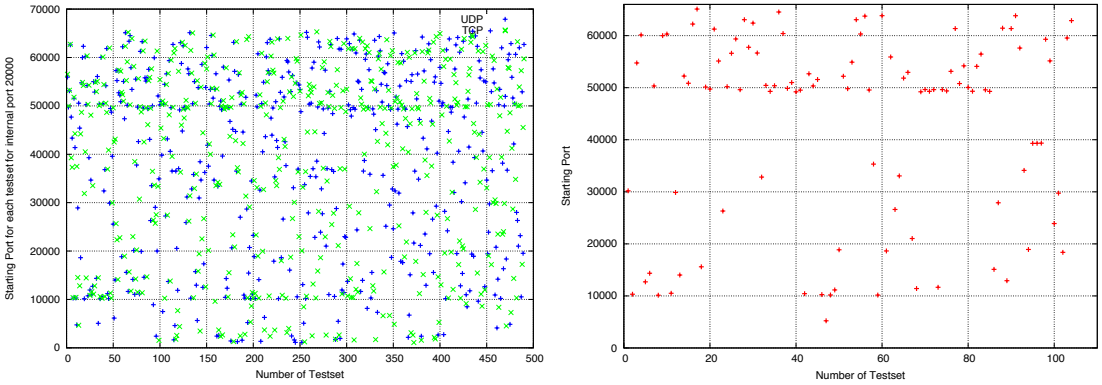

Fig. 4.33: Starting ports for the internal port 20,000. All non-preserving results are depicted on the left, the category fixed-delta on the right

The second category, **pattern-based** mapping, is depicted on the left hand side of figure 4.32. An iterative pattern with a defined period can be described as a sequence of deltas $(\Delta_1, ..., \Delta_n)$ dependent on the starting port *sPort*. Again we differentiate between an independent and dependent mapping and need to take care of the 16bit port number limit by defining a wrap-around port. Detecting patterns is not trivial since the period of the patterns differ. If too little testruns are conducted, the pattern may not be completely identifiable.

Finally, the last category as depicted on the right hand side of figure 4.32 is called **error-prone** and can also be divided into independent and dependent mapping. The blue resultset shows a fixed-delta with some outliers independent on the source port. The green resultset however, shows a more or less unstructured and random mapping strategy. By calculating the standard deviation of all the values and by specifying the minimum and maximum it is possible to narrow down the predicted port.

We finally looked at the starting ports of each testrun that were assigned for our internal source port 20,000. Figure 4.33 shows the distribution for all testsets that use a non-preserving port mapping strategy on the left and the distribution for non-preserving mappings that also use a fixed-delta strategy on the right (for UDP). In general no cluster can be identified, but the starting ports for fixed-delta mappings are most likely within the 50,000 range. Additionally, none of the fixed-delta mappings use well-known ports as their external ports.

Finding 3: *By analyzing port mapping patterns it is possible to predict an external mapping independent of the implemented binding strategy. Our identified categories allow predicting external mappings, either directly or by providing an interval and a probability for the error.*

4.6.4 Additional Behavior Results

Additional behavior measurements cover the *Stateful* (*StateTimer* and *StateTable*) and the *Filtering:Protocol-based* element of our information model.

Timeout Results

Initially our goal was to determine mapping timeouts for UDP and for TCP in the established state as described in section 4.3.1. In order to keep the duration of the test as short as possible our intention was to only test for TCP timeouts smaller than

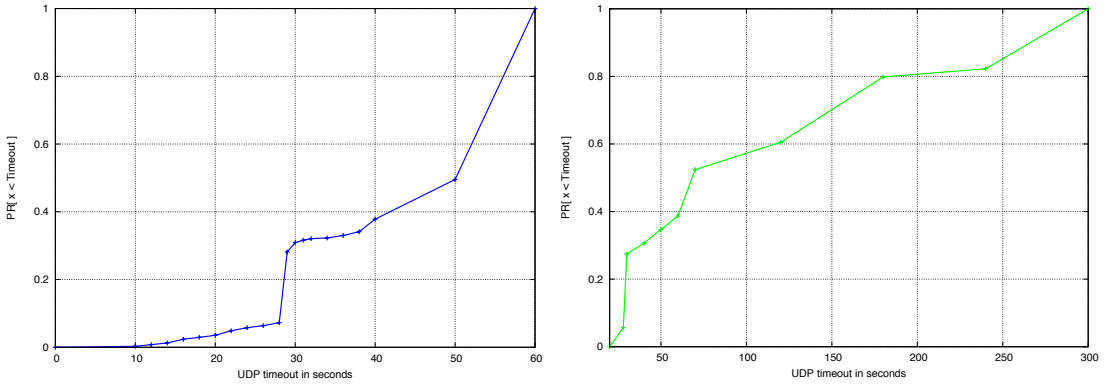

Fig. 4.34: CDF of the UDP timeout measurements

5 minutes. However, this didn't give us enough results to actually make a statement about TCP timers. Experiments in the lab have confirmed results of [68] and [60] and therefore TCP timeouts were not considered any further.

When focussing on UDP timeouts we conducted two measurements. The first one with a cut-off time at 60 seconds was included in a regular testrun to minimize the waiting time for the participant. A second UDP timer test was part of our topology measurements as described in section 4.3.4. Figure 4.34 shows the results as a CDF. For testset 2 (on the left) less than 3.5% implement a UDP timeout smaller than 20 seconds. A very popular timeout is 30 seconds and was found in more than 20% of our testset 2 and more than 25% of the topology test results (the diagram shows the last successful test, thus $x = 29$). 12% implement a timeout between 40 and 50 seconds and more than 50% a timeout greater than 60 seconds. The cut-off value for the topology tester was 300 seconds. Our results comply with the results of [68] where a mean value of 160 seconds was measured.

Mapping Table Results

When trying to find the table size and strategy belonging to the *Stateful:StateTable* element, the test client has to establish the maximum number of connections a stateful middlebox is able to handle at the same time. The first problem occurred because of the operating systems limitation of 1024 sockets per process, which means an implementation has to fork many processes and maintain them. This is not possible without using the JAVA Native Interface and therefore not applicable to a web-based measurement.

We included the algorithm in the linux-based version of testset 1, but after a few tests we received complaints from participants about crashing their router when approaching the limit. In fact, some routers actually stop working when the maximum number of mapping entries is reached and the only way of getting them to work again is to do a hard reset. A second problem was on the server side. When serving multiple clients at the same time, scalability issues were discovered. Therefore, we decided not to include the mapping table algorithms in our field test. However, measurements in the lab showed that the size of the mapping table has been raised on new devices lately. While older versions of OpenWRT set the */proc/sys/net/ipv4/netfilter* value to 8192, the most current version already sets it to 16384. Other common (older) devices such as the D-Link DI-524 set it to 4096, while our Netgear RP614v4 router is able to maintain approx. 6000 entries. All of the tested devices block new connections when reaching the maximum number of connections and don't override existing ones. Finally, the default value for *iptables* in most Linux distributions (e.g. Ubuntu and Debian) is 65536.

However, the number of available entries is still important when considering today's applications. For example, a single session with Google's Map service[17] established approx. 30 to 50 TCP connections simultaneously. Thus, when sharing a home router among an average size family the number of maximum connections may still be enough, but for a provider that wants to deploy Large Scale NAT serving hundreds of customers, the maximum number needs to be adjusted accordingly.

ICMP Results

ICMP messages are sent as a response to transport layer packets to signalize errors in the communication path. In order to map between the ICMP message and the corresponding transport layer packet the ICMP message carries parts of the network and transport layer header in its payload. We are interested in how ICMP packets are actually handled by middleboxes and if it is possible to modify the payload in order to embed additional information (as used for our autonomous middlebox traversal algorithm as published in [100] and described in section 6.5.2). Table 4.7 shows the results of our measurements.

Packet Sequence	Payload	Success Rate
Outgoing Measurements		
ICMP TTL exceeded out	original	6.02%
	modified	5.87%
ICMP echo request out	original	92.58%
	modified	89.01%
ICMP echo reply out	original	8.03%
	modified	6.82%
Packet Sequences		
UDP out, ICMP TTL exceeded in	original	78.21%
UDP out, ICMP TTL exceeded in	modified	65.1%
echo request out, echo reply in	original	91.98%
echo request out, echo reply in	modified	84.81%
echo request out, TTL exceeded in	modified	48.3%

Tab. 4.7: Protocol-based filtering of ICMP packets of a subset of testset 1

For outgoing tests, we were looking for those ICMP packets that can be sent to the external network without requiring them to be part of any other communication. An ICMP echo request message was allowed in 92.58% (89.01% for modified packets) since this message is used to initiate a "connection" according to the protocol. When trying to send out ICMP TTL exceeded messages we were only successful in 6.02% (5.87% modified). This is because such messages are normally only sent as a response to a transport layer packet and not as a packet initiating a "connection". The same is true

[17] http://maps.google.com

for ICMP echo reply messages. To summarize, almost 95% of all tested middleboxes implement a protocol-based filtering strategy for ICMP.

When looking at incoming packets, we first sent out a regular packet (either a UDP packet or an ICMP echo request packet) and tested if the ICMP packet sent in response reaches the test client. 91.89% allowed "pinging" an external host and 78.21% forwarded TTL exceeded messages as a response to UDP packets. Finally, modifying the payload of ICMP packets influences the success rate for traversal only marginally.

Finding 4: *Independent on the analyzed property, middlebox behavior differs from model to model. The size of the mapping table is not critical unless LSN is deployed. ICMP error handling mostly works, but is not implemented correctly. We recommend to assume a UDP timeout smaller than 30 seconds in order to avoid problems with many implementations.*

4.6.5 Behavior-based Traversal Results

As described above we implemented several versions of common hole punching algorithms and tested to which extent they work with existing middleboxes. The success rates only show the percentages for a constellation where only one host (the service) is behind a stateful middlebox since our test server is located in the public Internet. When also considering different middlebox constellations as in section 4.6.2, the percentages depend on the behavior of both middleboxes.

Traversal Technique	**Testset 1**	**Testset 2**
UDP Hole Punching with high TTL	68.09%	73.87%
UDP Hole Punching with low TTL	n/a	58.49%
UDP Hole Punching Silent	n/a	76.91%
TCP HP low	40.82%	62.33%
TCP HP high	31.77%	52.46%
TCP SILENT	n/a	45.61%
TCP FTP ALG	31.35%	n/a
UDP combined	68.09%	77.92%
TCP combined	65.03%	66.5%
Traversal combined	83.88%	84.57%

Tab. 4.8: Results of the traversal measurements for testset 1 and 2

UDP hole punching with a high TTL, and therefore provoking ICMP port unreachable messages as a response to the initial UDP hole punching packet, was successful in 68.09% for testset 1 and 73.87% for testset 2. When setting the TTL field of the IP packet to a value lower than the number of hops between the test client and the server, an ICMP TTL exceeded message is sent as a response to the hole punching packet. This technique was successful in 58.49%. Finally, when emulating a middlebox that does not send ICMP port unreachable messages as a response to the hole punching

packet, the traversal was successful in 76.91%. For TCP the results are different. When setting the TTL of the hole punching packet to a low value, the success rate is significantly higher than with a high TTL (62.33% vs. 52.46% for testset 2). Determining an appropriate TTL value can be done using our topology measurement algorithm as described in section 4.3.4.

When considering any traversal method for UDP a combined success rate of 77.92% was observed, which is very close to the 78.45% that implement an endpoint independent binding. Thus, UDP hole punching works in 99.46% for all endpoint independent middleboxes if a technique is applied that also considers the behavior of the middlebox. For connection dependent middleboxes our findings in section 4.6.2 can be applied for developing new port prediction algorithms that will work for approx. 60% of all connection dependent implementations. Although TCP is a complex protocol compared to UDP, we still reach a success rate of 88.37% for endpoint independent middleboxes when carefully selecting the technique according to the behavior.

Finding 5: *A single traversal mechanism can never be as effective as a solution that carefully parameterizes a basic traversal algorithm according to the current situation, the topology and the behavior of the involved middleboxes. This allows increasing the success rate for UDP from 58.49% to 77.92% and for TCP from 45.61% to 66.5% (best case).*

4.6.6 Additional Results

Our additional measurements cover the protocols UPnP, SCTP, as well as a test for different Application Layer Gateways and for an image proxy. For UPnP we used the MiniUPnP library[18] for testset 1 and the Cling library[19] for testset 2 since MiniUPnP seems to have problems with many implementations. Thus, a UPnP capable device was found in 30.59% of testset 1, but we were only able to actually utilize the Internet Gateway Device (IGD) in 21.73%. With Cling, we found a UPnP enabled model in 37.16% and were able to query the external IP address for 33.14%. Thus, approximately one third of all middleboxes of testsets 1 and 2 actually enabled UPnP.

When testing for the support of the transport layer protocol SCTP [146], we used the user-space SCTP library for Windows[20] and for Linux the libsctp library that is included in all common distributions. Our results show a success rate of 15.87% for establishing a SCTP association. Similar results (for 34 different middleboxes) were published in [68]. Most of the middleboxes that support SCPT are Linux-driven (e.g. OpenWRT-based firmwares) since a SCTP module for *iptables* exists. We didn't test for the support of DCCP [85] since no stable implementation for Windows existed at the time we conducted our tests. However, none of the 34 NATs tested in [68] supported DCCP.

Application Layer Gateways for the SIP protocol [131] were found in 24.17% and 78.28% also support the translation of FTP (results of testset 2). Finally 1.12% of testset 1 (mostly mobile operators) implement a proxy that manipulates JPEG images by shrinking their size. We did not conduct further tests on content manipulation, filtering and censoring, but leave this interesting and manyfold topic for future work.

[18] http://miniupnp.free.fr/
[19] http://4thline.org/projects/cling/
[20] http://www.sctp.de/

4.6.7 Topology Results

The architecture of our measurements is based on a test client located in the participant's network and a test server in the public Internet. Thus, the whole topology between the client and the server is treated as a blackbox since individual middleboxes cannot be detected with the state of the art. In most cases a middlebox is only deployed as a CPE (Customer Premises Equipment), usually one hop from the test client. However, additional firewalls, double NAT at the user side, Large Scale NAT at the provider side or proxies deployed in the network might influence our results. There are three possibilities to detect multiple stateful middleboxes with our experimental results: First, by comparing IP addresses of the UPnP results with the actual public IP addresses, second by our topology detection algorithm as presented in section 4.3.4 and third by analyzing the IP addresses of our Android-based test clients.

Once a participant finishes a testrun (web-based client, testsets 1 and 2) the results are committed using a HTTP-POST. By comparing the source IP address of this packet with the external IP address that is found in UPnP, we have a first indicator for double NAT. 24% of all UPnP enabled devices carried an external UPnP address that was different from the one that was used to post the results. In 24.5% of these cases the UPnP address came from the 10/8 range, in 3.6% from the 172.16/16 range, in 58.3% from the 192.168/24 range and in 13.6% the external UPnP address was also a public one.

Our algorithm as described in section 4.3.4 allows detecting stateful middleboxes on the path from the test client to the Internet. The UDP-based implementation requires that ICMP messages are generated and forwarded by intermediate nodes. However, in many cases where stateful middleboxes are involved, the provider blocks outgoing ICMP messages as an answer to incoming UDP packets, which leads to problems with our algorithm. Therefore, we were not able to reveal the exact topology for many results. A TCP-based implementation using RST packets is highly recommended for future work although it requires superuser privileges on the client side. However, the proposed method is still valid for estimating the approximate position of the LSN and for determining a valid TTL value for hole punching (which will be further described in the following chapter). We found LSNs to be present in fixed-line networks (e.g. the German provider EncoLine[21] most likely implements a Linux-based LSN), as well as in LTE networks that provide Internet access to remote areas. In the following we show the topology of the Vodafone Germany LTE network that could be revealed using our test. The resulting table from our detection algorithm is shown in table 4.9.

	Hop 11	**Hop 12**	**Hop 13**	**Hop 14**	**Hop 15**	**Hop 16**	**Hop 17(C)**
20s	17	17	17	17	17	17	17
30s	16	16	16	16	16	16	17
50s	16	16	16	16	16	16	17
60s	16	16	16	16	16	16	17
90s	13	13	13	16	16	16	17
120s	13	13	13	16	16	16	17

Tab. 4.9: Result for the Vodafone Germany topology

[21] http://www.encoline.de/

For a timeout value of 20 seconds, the test server is able to reach the client (hop 17) no matter which mapping the client tried to remove. Please note that removing a mapping for UDP means that the TTL value of the servers keep-alive packets is set to $n-1$, n being the number of hops as seen by the client. For a timeout of 30s the test server is able to reach hop 17 only if the mapping of hop 16 has not been removed. Thus, hop 16 implements a stateful middlebox with a timeout value between 20 and 30 seconds. The same is true for hop 13 that implements a stateful middlebox with a timeout between 60 and 90 seconds. The server is able to reach beyond hop 13 only if either the timeout has not triggered yet (here 60s) or if the mapping has not been removed yet (columns hop 14 to hop 17). With this table we are able to generate the corresponding topology as shown in figure 4.35.

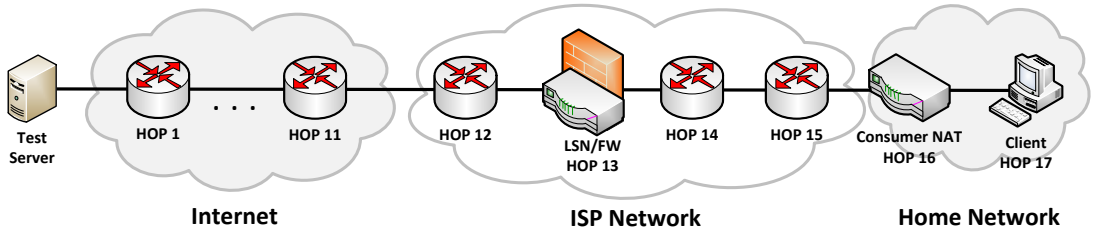

Fig. 4.35: Revealed topology for the Vodafone Germany LTE network

Altogether, Large Scale NATs and other stateful middleboxes are already deployed by many providers and cause additional problems for middlebox traversal. When looking at our preliminary Android-based results, all mobile operators issue private IP addresses to their clients, a promising result for future research. To reveal further topologies we propose to extend our algorithm by a TCP-based version. New prediction algorithms might furthermore allow to predict approximate positions of stateful middleboxes.

4.6.8 Lessons Learned

The last sections presented the results of our field test based on our information model and based on the algorithms that we have developed for experimentally analyzing middlebox behavior. Many efforts have been made to classify the behavior of middleboxes and to develop solutions for coping with them, but the state of the art does not have a satisfying answer to the question why middleboxes still introduce many problems to today's networks and why existing techniques don't work as expected. The results of our field test shed light on these questions and serve as a basis for our solution to the middlebox traversal problem that will be presented in the following chapters.

We started our field test by running the popular STUN algorithm and found a Port Address Restricted Cone NAT to be the most popular one (66% of all tested middleboxes). Symmetric NATs, which are said to introduce many problems, were found in 8%, but when only considering the most current results (testset 2), the number increased to 19%. These number were a first warning for us and we conducted a more detailed test on the binding behavior, which is part of the *Translation_and_Modification* element of our information model. We found 78.45% for UDP and 75.25% for TCP to implement an independent binding strategy, which means that predicting an external port is possible by querying an external server (such as STUN) as done in the state of the art. However, when comparing the results to the STUN results, we also found that the STUN classification only produces correct results for Restricted Cone NATs, but fails for

Field Test Results	Recommendation	Expected Results
1) SotA: STUN is used to classify middlebox behavior.		
Approx. 25% of FC wrong. 5% of SYM wrong.	Strict separation between predictable and unpredictable endpoints.	Clear statement about applicability of traversal techniques.
2) SotA: Middlebox traversal often fails for two hosts both behind NAT.		
Only 61.4% (56.6% for TCP) of all constellations work with the SotA.	Also consider constellations with connection dependent mapping and swap roles if necessary.	68.4% (63.4% for TCP) of all constellations will work.
3) SotA: Middleboxes with connection dependent bindings are hard to traverse.		
22% (25% TCP) implement a connection dependent binding. For 57% (44% TCP) we found predictable binding patterns.	Implement port prediction based on binding behavior.	Ports can be predicted for 91% UDP and 86% TCP.
4) SotA: Hole Punching works in some cases and doesn't in others.		
Success rate for a single variation: between 58.49% (45.61% TCP) and 76.91% (62.33%).	Parameterize hole punching based on the behavior of the involved middleboxes.	77.92% for UDP and 66.5% for TCP.
5) SotA: Large Scale NAT has only been considered marginally [151].		
Large Scale NAT is already deployed by providers in numerous networks and influences end-to-end communication, esp. in mobile networks.	Take topology into account when making decisions about middlebox behavior.	Allows implementing lowTTL HP with LSN: 62.33% vs. 45.61% for TCP (best case).

Tab. 4.10: Lessions learned

Symmetric NATs (in 3.85% for UDP and 9.79% for TCP) and Full Cone NATs (21.94% for UDP and 34.18% for TCP). Thus, we follow and confirm the recommendation of [130] not to use the STUN algorithm for categorizing middleboxes, especially not for NAT binding behavior. However, for classifying filtering behavior, STUN still produces correct and valuable results.

With these findings we were able to predict the success rate of behavior-based traversal as used in the state of the art. Hole punching only works if both hosts are able to predict their external port, which only 61.5% (56.6% TCP) of our constellations are capable of. Considering the fact that state of the art hole punching works with 90% of all endpoint independent bindings, only 55.4% of all constellations where both hosts are located behind a middlebox will be able to successfully establish a direct connection. As TCP is more complex than UDP the situation is even worse. In 56.6% both hosts are endpoint independent and the state of the art traversal using a low TTL for the hole punching packet showed a success rate of 62.33%, which is 88.37% of all endpoint independent middleboxes. Thus, TCP traversal for an arbitrary constellation works in approx. 50%.

When looking at the individual results of our field test we found a large window of opportunity for improving these numbers. First, identifying patterns in connection dependent bindings also allows predicting the external port of approx. 60% (46% TCP) of all connection dependent middleboxes. Second, taking into account that middlebox constellations where only one end implements a non-predictable binding strategy can also be traversed by considering the behavior of both ends, we recommend to choose an appropriate technique for the constellation and swap roles if necessary. Third, predicting external mappings by analyzing mapping patterns helps to further increase the probability for establishing direct connections through middleboxes. Fourth, by parameterizing existing hole punching techniques it is possible to increase the success rate even further. And finally, by also taking the topology of a path into account it is possible to cope with the increasing number of middleboxes that are operated by ISPs.

Altogether, the state of the art has provided numerous methods and techniques for coping with middleboxes, but due to the non-standardized behavior their success rate depends on the situation. Our field test shows that a single mechanism, such as one variant of hole punching, can never be as effective as a solution that takes the topology, the behavior of involved middleboxes, as well as further user-defined requirements into account. Only if such solutions exist, middleboxes may become a valid design principle for the future Internet.

4.7 Summary and Key Findings

In this chapter we showed that a thorough and structured analysis of middlebox behavior helps to improve the success rate of middlebox traversal techniques (Q1 according to section 1.1). We presented an information model to describe middlebox behavior in a structured way. Different measurement algorithms were then developed targeting the individual elements of the model. After evaluating the algorithms in our virtualized testbed that supports emulating arbitrary middlebox behavior, a public field test was conducted. As a result, we are able to categorize middlebox behavior into “good” and “bad” behavior and our findings help to better understand existing deployments. Based on our experimental analysis we can give recommendations how to improve existing traversal techniques and argue that an optimal solution for coping with middleboxes can only be found if their behavior is understood and a traversal algorithm is selected and customized accordingly.

Key Findings of this chapter
(and contributions according to section 1.2):

C4.1 **Our information model contains relevant middlebox properties and serves as a formal representation for middlebox behavior.**

C4.2 **The test routines and algorithms designed in this chapter allow systematically measuring middlebox behavior and to create an information model instance that represents a specific middlebox.**

C4.3 **An experimental analysis first verified our algorithms in a virtualized testbed, before conducting a public field test with the following findings:**

C4.3a **Only 68.4% (63.4% for TCP) of all constellations in the Internet provide the essential prerequisites for behavior-based traversal.**

C4.3b **Even if the middlebox implements a connection dependent binding strategy, in approx. 60% for UDP and 46% for TCP it is still possible to predict the external binding by analyzing binding patterns.**

C4.3c **Large Scale NAT is already deployed by many ISPs and hinders communication, especially in mobile networks.**

C4.3d **A single traversal mechanism can never be as effective as a solution that carefully parameterizes a basic traversal algorithm according to the current situation, the topology and the behavior of the involved middleboxes.**

Part III

TRAVERSAL OF MIDDLEBOXES

5. MIDDLEBOX TRAVERSAL AND SERVICE PROVISIONING

5.1 Introduction

Middleboxes have been introduced to IP-based networks to solve specific and in many cases also urgent problems. For example, by allowing many hosts to share one address, NAT fights against the lack of public IPv4 addresses. As an unwanted side-effect of this achievement the connectivity across NAT devices is affected. According to our information model as described in section 4.2, NATs, or stateful middleboxes in general, maintain a state table for all connections and allocate a timer or policy to each entry to define the lifetime of it. Packets traveling across the middlebox are then processed according to the state. The same is true for firewalls that were introduced as a network perimeter to prevent the communication of unauthorized applications across network domains. Administrators who add a firewall to the communication path see the blocking of certain applications as an intended feature. However, as network protocols get more and more complex and applications implement their connectivity logic on the application layer, firewalls may also block certain applications as a side-effect because they don't understand the details of the protocol. Furthermore, providing a globally accessible service from behind a stateful middlebox requires to cooperate with the middlebox in order to establish a state that allows incoming packets to be forwarded to the defined service. All these scenarios and problems are examples for the so-called "middlebox traversal problem".

The traversal of middleboxes has been heavily discussed in research and standardization, but the definition of middlebox traversal is rather vague. In general, middlebox traversal is seen as a technique to communicate across middleboxes, but in our opinion this is not precise enough. We follow the authors in [139] and state that a middlebox traversal mechanism should take further requirements into account.

In this thesis we argue that a middlebox traversal solution has to take the role of the application and the intention of the deployment of the middlebox into account. For example, if a middlebox breaks the connectivity between two hosts only as a side-effect, the traversal technique has to make sure that applications still work across the middlebox as if the middlebox didn't exist. On the other hand, if an administrator aims to restrict the access to certain services, applications or protocols, the traversal technique has to follow this policy. We thus define middlebox traversal for our thesis as follows:

Definition: *Middlebox Traversal describes a protocol or mechanism that enables an application to communicate across middleboxes in a way that it is compliant with authorization and accessibility requirements, as well as additional policies as defined by the user or the administrator of the network.*

For example, if users behind a consumer NAT simply want their application to work across all middleboxes on the path, the utilized traversal technique can be chosen only based on criteria such as performance. However, if a user wants to provide a service to

specific external hosts only, the traversal technique should take this into account. In enterprise environments administrators are usually not thrilled about middlebox traversal techniques, because they violate their security policy. Thus, it is important that a middlebox traversal mechanism also considers policies as defined by (super)users. In addition to authorization requirements, we have to consider the availability of external infrastructural components and what the impact of the infrastructure could be. For example, external data relays operated by a third party might violate security and privacy policies of a user and should therefore be avoided for critical traffic, while still being a legitimate choice for already encrypted traffic.

The following sections present the middlebox traversal problem in more detail. We then discuss the state of the art in middlebox traversal, which leads to the definition of new service categories for middlebox traversal. In the following chapter we then present our knowledge-based approach that takes many variables into account when selecting the most suitable technique for middlebox traversal.

5.1.1 Middlebox Traversal Problem

As stated in chapter 3.2 when presenting our middlebox processing model, a middlebox operates between two network realms by modifying, filtering or adding packets that are seen on its network interfaces according to middlebox specific processing rules. The middlebox traversal problem occurs when packets are intended to be processed, but the middlebox does not have the appropriate processing rules. Based on figure 5.1, this section presents four problem categories for middlebox traversal that are derived from [71]. The figure depicts two packet flows, one from the source to the destination and another flow from the destination to the source. The middlebox implements packet translation and thus two endpoints for each packet flow exist: a local endpoint describing the combination of an IP address and transport layer port for the private realm, and a combination for the public domain towards the destination. For a regular middlebox that translates packets and operates in a way that sessions are established from the local to the public domain, the internal, as well as the mapped endpoints of the two packet flows are identical. Dependent on the problem category, either one packet flow is involved or the endpoints differ in a way that they are not related and therefore cause problems.

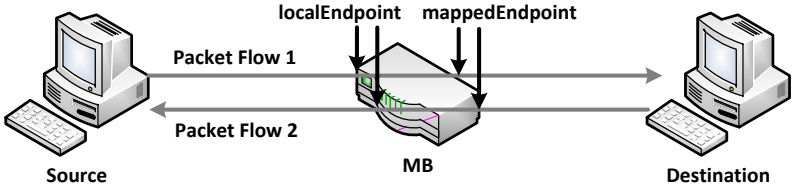

Fig. 5.1: Illustration of packet flows for the middlebox traversal problem

Realm-specific Addresses

For the first problem category the packet flow from the source to the destination (flow 1 according to figure 5.1) contains information that is essential for the corresponding protocol, but the middlebox is not able to handle it. This is the case if a protocol carries **Realm-Specific Addresses** in its payload, such as the Session Initiation Protocol (SIP) [131] that uses IP addresses and transport layer ports within its payload to

signalize where related packets should be sent to. If the middlebox is not able to translate these packets using an Application Layer Gateway (ALG), the protocol is most likely to fail.

```
1 if found (pub := get_state(p)) then
2 |   p := translate_packet(p, pub);          ▷ translate packet immediately
3 else
4 |   pub := allocate_new_mapping(p);         ▷ a new external mapping is needed
5 |   store_state(p);                          ▷ also remember layer 7
6 |   p := translate_packet(p, pub);           ▷ also translate layer 7
7 end if
8 send(p, external);                          ▷ call send function in output stage
```

Alg. 5.1: $Proc(internal, p)$ for Realm-specific address processing

The second packet flow according to figure 5.1, which is logically related to the first one, will be sent to a wrong (local) endpoint since the addresses in the payload of packet flow 1 are not translated. According to our information model, neither the *Translation_and_Modification* element nor the *Filtering* and *Stateful* elements support the correct (application layer) protocol. According to algorithm 5.1, the middlebox would have to set the correct state entry in step (2) (for packet flow 2) and translate the application layer protocol in step (3) in order to allow the incoming packet flow to traverse the middlebox.

Peer-to-Peer Applications

The second problem category, *Peer-to-Peer Applications*, targets hosts behind a middlebox that offer a globally reachable service. This is not only true for traditional services such as a web-server, but also for applications that require a direct connectivity between two arbitrary hosts. For example, Voice over IP uses direct connections between the caller and the callee for transferring packets containing voice. According to figure 5.1 this means packet flow 1 does not exist and packet flow 2 is initiated by the destination. Since stateful middleboxes only establish state on outgoing packets the incoming packet flow 2 is not related to any outgoing one and cannot be forwarded.

```
1 if found (priv := get_state(p)) then
2 |   do_something();                          ▷ this will never be reached
3 else
4 |   drop_packet(p);                          ▷ state not known: drop packet
5 end if
```

Alg. 5.2: $Proc(external, p)$ and the Peer-to-Peer Application problem

More precisely, algorithm 5.2 shows the processing of incoming packets according to our processing model for the P2P applications problem category. The middlebox receives the packet on the external interface and queries the state table for an existing entry. Since no entry is found the packet is immediately dropped.

Bundled Session Applications

The third category, *Bundled Session Applications*, is a combination of the other two. Bundled session applications, such as FTP [120] or RTSP [138], carry realm-specific addresses and ports in their payload to establish a second session. The first session is usually referred to as the control session (packet flow 1), while the second session (packet flow 2) carries the actual data. The problem here is not only the realm-specific addresses, but also the data connection of a bundled session application that is established from the public realm towards the private one, a direction the stateful middlebox does not permit. According to figure 5.1, the second packet flow is logically related to the first one, but sent to a different mapped endpoint. If the middlebox does not understand this relation, the second flow will be blocked as with P2P Applications. The solution of the traversal of bundled session applications is a combination of the solutions as presented above: An ALG needs to understand and translate the application layer protocol and a state entry needs to be added to the mapping table.

Unsupported Protocols

In addition to the three problems above we identified the last category, *Unsupported Protocols*, which includes non-translatable (flow 1) applications and protocols. Newly developed transport layer protocols, such as the Stream Control Transmission Protocol (SCTP) [144] and the Datagram Congestion Control Protocol (DCCP) [85] cause problems with middleboxes even if an internal host wants to establish a connection to a public one. This is because current middleboxes do not have built-in support for these protocols. In case of SCTP with its multi-homed design, it is not trivial to adapt translation and modification support from UDP and TCP. As described in [154], it is not possible for the SCTP checksum to be recalculated only based on the difference to previous packets. Therefore, traversing a legacy NAT with SCTP-support would result in a significant packet delay. There are already suggestions to the problem [144] that propose changing the NAT, which is not possible in most cases. Unsupported protocols also cover protocols that cannot work with middleboxes because their layer 3 or layer 4 header is not available for translation. This happens when using security protocols such as IP-Security (IPSec) [83].

5.2 State of the Art in Middlebox Traversal

Over the past years a large number of traversal methods emerged in research and standardization. The following sections give a detailed overview of the state of the art in middlebox traversal. First, section 5.2.1 introduces techniques that need to be explicitly supported by the middlebox. Behavior-based approaches are presented in section 5.2.2. Section 5.2.3 covers further solutions. Finally, section 5.2.4 compares existing solutions and summarizes our findings.

5.2.1 Explicit Solutions

Static Port Forwarding and Universal Plug and Play

The most intuitive way to allow an incoming non-related packet flow to traverse the middlebox is to statically add an appropriate entry to the state table. Most modern routers provide a web-based interface and pre-configured rules for popular services. The drawback of this solution is not only the limited flexibility due to a static approach,

but also that it requires basic network knowledge in order to access the router and to forward the correct ports.

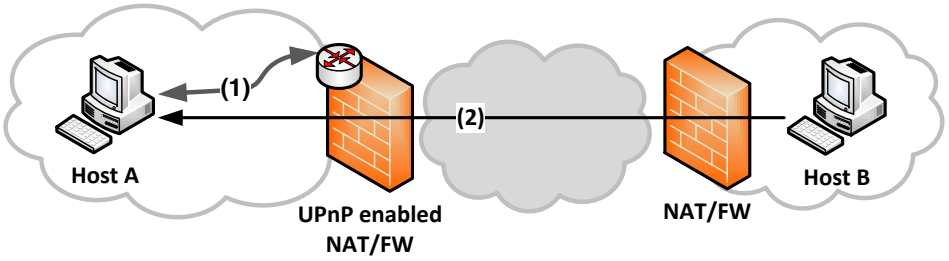

Fig. 5.2: Univeral Plug and Play (UPnP) and NAT-PMP

Universal Plug and Play (UPnP) is a set of protocols first proposed by the Microsoft Corporation and now under the control of the UPnP Forum [50]. UPnP enables the seamless communication between all devices in the network and allows controlling networked devices such as printers, multimedia devices, as well as home routers that implement an Internet Gateway Device (IGD). This enables applications to dynamically add, remove and control port forwarding entries to an UPnP-enabled middlebox. Since UPnP does not come with any security mechanisms, many routers are shipped with UPnP disabled. Instead of using UPnP, section 6.5.1 proposes a secure IGD based on the Devices Profile for Web Services (DPWS) [42]. A similar protocol, the NAT Port-Mapping Protocol (NAT-PMP) is mainly pushed by Apple and standardized in the IETF [22].

Next Steps in Signaling: NSLP

The NSIS Signaling Layer Protocol (NSLP) for NAT and firewalls [147] is a signaling protocol that allows configuring middleboxes on the path between two hosts. The basic assumption of the RFC is that the sender and the receiver, as well as all involved middleboxes are NSLP aware. By using two messages, *create* for outgoing flows and *external* for incoming data flows, NSLP enables hosts to configure NATs and firewalls according to the expected data flow and policies of the network. For example, a receiver that is expecting incoming packets may use an *external* message to direct all stateful middleboxes that would otherwise block the unknown packets due to missing state to process and forward them. This also allows the traversal of multi-layered middleboxes as long as all of them support the protocol. The RFC also discusses security issues and recommends to use authentication and key exchange mechanisms as defined in [137].

MIDCOM: Middlebox Communication

Middlebox Communication (MIDCOM) [143] aims at shifting application specific functionality from a middlebox itself into separate MIDCOM agents. Agents assist middleboxes to process and traverse applications that in general have problems with middleboxes because of realm-specific addresses or due to bundled sessions as described above. More precisely, MIDCOM agents implement Application Layer Gateways (ALG) that are independent of the core middlebox itself and are able to support an application layer protocol. MIDCOM agents are controlled by the MIDCOM protocol that can be used to configure and deploy application specific rules.

NUTSS

The NUTSS (NAT,URI, Tunnel, SIP and STUNT) architecture is an novel approach to connectivity establishment. NUTSS was first mentioned in [63] and more thoroughly documented in [61]. The basic idea is to use globally unique URIs for endpoints instead of IP addresses. If a user wishes to establish a connection, he first initiates an end-to-end signaling. Once a connection request is authorized by the end-hosts as well as all middleboxes on the way, necessary state is established and a working 5-tuple is provided to the initiator. NUTSS has two main components, policy boxes (P-boxes) and middleboxes (M-boxes). P-boxes form an overlay and are used for signaling a connection request. After signaling, M-boxes are configured according to the policy and used to enforce this policy for the actual data connection. NUTSS also proposes to include traversal techniques such as STUN(T) and proposes new ways of establishing connections via TCP, which will be further described in section 5.2.2.

Application Layer Gateway

An Application Layer Gateway (ALG) operates on the application layer and helps middleboxes by transparently translating application specific details such as realm-specific addresses. ALGs for protocols such as FTP and SIP are often found in consumer grade middleboxes and may also help to create the correct state entries for bundled session applications.

SOCKS

SOCKet Secure (SOCKS) [90] is a protocol to use with a proxy server that resides on the edge between two domains. SOCKS provides two modes of operation: *connect* directs the proxy to connect to an external endpoint, while *bind* is used for bundled session applications in order to allow secondary connections towards the client. Many operating systems and applications support the SOCKS protocol, but only a few home routers actually implement it for middlebox traversal.

RSIP

Realm Specific IP [12, 13] is considered an alternative to Network Address Translation and allows leasing public endpoints to RSIP hosts within a private domain. The public endpoint may be a unique IP address and a port or a shared IP address and a number of unique ports. A RSIP client uses this allocated endpoint for communicating with hosts in the Internet and tunnels them through the private realm. The RSIP gateway, usually implemented on the middlebox that is translating addresses between the realms, decapsulates the packets and sends them off into the Internet.

5.2.2 Behavior-based Solutions

STUN

Rather than being a middlebox traversal technique, Simple Traversal of User Datagram Protocol through NATs (STUN) [133] and its successor Session Traversal Utilities for NAT (also STUN) [130] is a "tool that is utilized as part of a complete NAT traversal solution" [130], such as ICE [128] and SIP-Outbound [77].

STUN defines two main messages that allow discovering the public IP address and port of an internal mapping: binding requests and binding responses. As depicted in

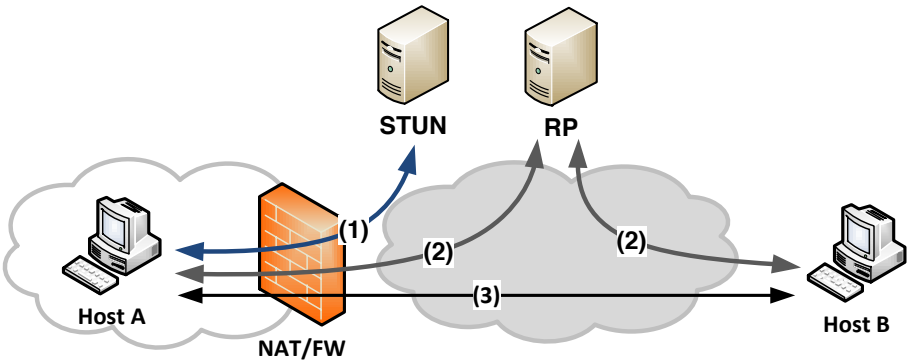

Fig. 5.3: Session Traversal Utilities for NAT (STUN)

figure 5.3 a client starts by sending a binding request (as a UDP packet by default) to a STUN server which then replies with a binding response containing the outermost IP address and port in the *MAPPED-ADDRESS* field (step 1). This address can then be embedded into an actual traversal technique or used as a non-realm-specific IP address in the payload (2) to eventually establish a direct connection to a remote host (3). STUN only works with an endpoint independent binding strategy since only then the external mapping is dependent on the internal one. In case of a connection dependent binding the determined external endpoint is useless since it has nothing to do with the actual endpoint that will be used for the actual protocol that requests middlebox traversal. The initial STUN RFC, "Classic STUN", also defines four categories of middlebox behavior as described in section 3.3.3 and proposes an algorithm for discovering them. As already mentioned in section 4.6.2, the categories were removed from the most current RFC since they were found to be faulty. Today, the STUN protocol is mainly used by VoIP applications for replacing realm-specific addresses on the client instead of relying on the middlebox to do so. Skype[1] is also supposed to implement a STUN-like protocol for discovering external mappings.

UDP Hole Punching

STUN is also used as a basis for hole punching as depicted in figure 5.4. The learned external mappings are exchanged via a rendezvous point (RP) by embedding them in a signaling protocol (step 1). Once both endpoints have received the others mapping they start sending packets to the remote address. The packet sent in step (2) creates an appropriate mapping in hosts A's middlebox, but is blocked by host B's middlebox since no mapping exists. However, the packet sent by host B in step (3) matches the created state and will eventually reach host A. Hole punching was first mentioned in [71] and more thoroughly documented in [48]. Today, hole punching for UDP is used by many proprietary protocols for instant-messaging, online-gaming and VoIP applications.

Unfortunately hole punching only works in some scenarios dependent on the behavior of the involved middleboxes. Most importantly, hole punching requires both hosts to be able to predict their external mappings for the actual connection by using a STUN server (for independent bindings) or by running a port prediction algorithm taking our findings of the last chapter into account. For port address restricted filtering the destination port of the packet sent in step (2) has to match the source port of packet (3) and vice versa. For address restricted filtering a matching IP address is sufficient.

[1] http://www.skype.com

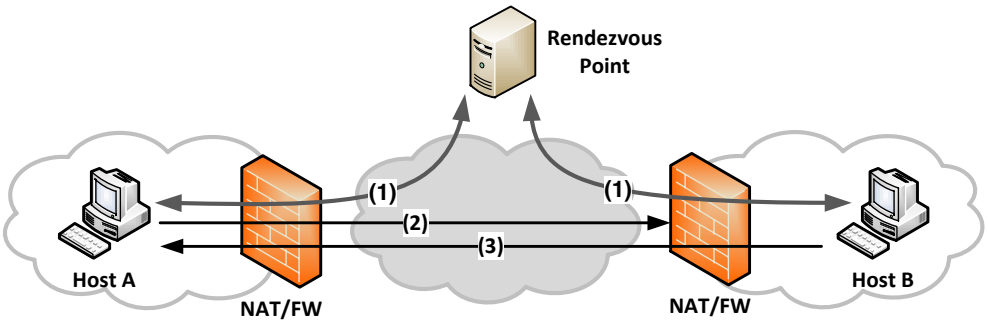

Fig. 5.4: Illustration of the hole punching algorithm

Finally, since both hosts may be located behind the same middlebox without knowing it, both hosts should also exchange their local mappings and send the packets of steps (2) and (3) to the local endpoints as well.

TCP Hole Punching

Due to the stateful design of TCP, TCP hole punching is not as trivial as for UDP. However, the basic concept is the same. Two hosts determine their external endpoints, exchange them via a rendezvous point and establish a connection with each other. The outgoing packets create a state in the state table which is then used by the remote packets for traversing the middlebox. In order to be able to establish TCP connections from the same source port to an external STUN-like server and to a remote endpoint, [48] proposes to use the *SO_REUSEADDR* option of the standard Berkeley Socket API (BSD sockets). This allows binding multiple sockets to the same source port, namely the session to the rendezvous point, the session to the remote endpoints (public and private), as well as one socket that is listening for incoming connections from the remote host. The exchanged messages are shown in figure 5.5.

A slightly different approach that does not require to simultaneously open TCP sockets is proposed by [11] and depicted in figure 5.6. Their *NATBLASTER* [11] framework uses TCP-SYN packets with low TTL values to create the mappings. The SYN packets are discarded in the network and will not reach the remote host. A third party is used for coordinating sequence and acknowledgement numbers and an appropriate SYN-ACK packet is forged using RAW sockets. The authors also focus on different scenarios and propose to use the birthday paradox to limit the number of tries in case of an unpredictable mapping.

The last two approaches as proposed in [60] and [47] have already been explained in section 4.3.2. Figure 5.7 shows the high TTL approach for two hosts both behind a stateful middlebox. Here, H1 punches the hole by sending a TCP-SYN packet to the remote endpoint and receives a TCP-RST as an answer. If M1 does not close the mapping immediately, the TCP-SYN packet of H2 establishes a direct connection between H1 and H2. Figure 5.8 shows the same approach for low TTL values. The initial SYN packet gets discarded in the network and triggers an ICMP TTL exceeded packet. The success rate of these approaches is highly dependent on the behavior of the middlebox, in particular if SYN-SYN sequences are allowed and if a TCP-RST or ICMP TTL exceeded packet closes a mapping or not. [60] provides a good comparison of the techniques and lists the issues and requirements (network/implementation) of each approach.

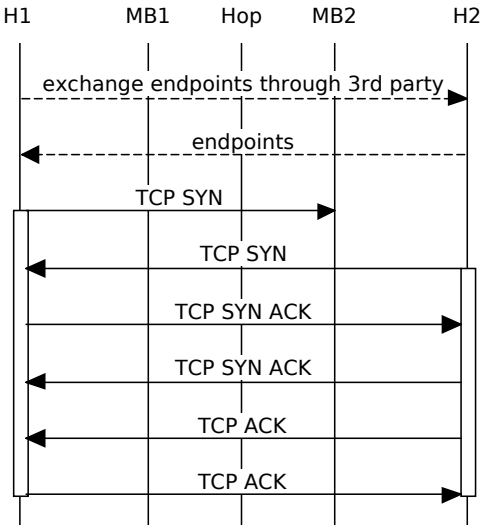

Fig. 5.5: P2P Hole Punching

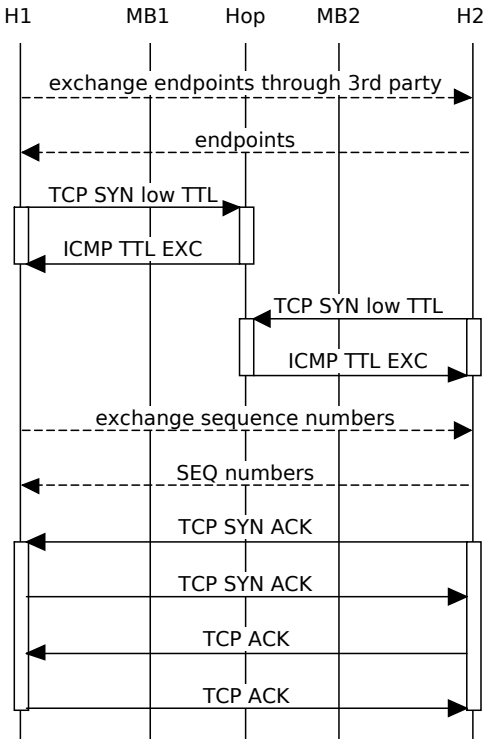

Fig. 5.6: NATBLASTER

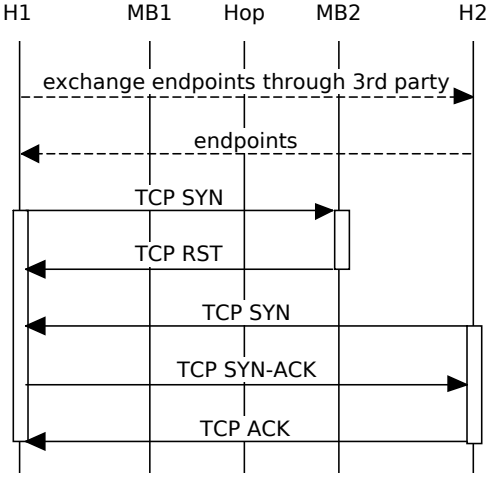

Fig. 5.7: STUNT using high TTL

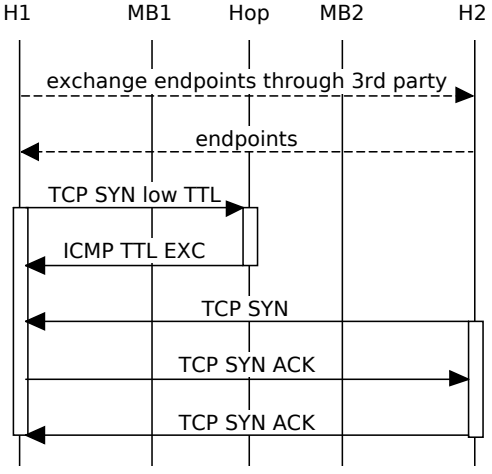

Fig. 5.8: STUNT using low TTL

Port Prediction Techniques

Translating middleboxes assign new mappings to each outgoing session according to their behavior as described in the last chapter. Port prediction is a technique that anticipates an external public mapping for a given private one. This is especially useful for middleboxes that implement a connection dependent binding strategy and cannot use STUN to query external mappings. The complexity of a port prediction algorithm depends on the way a middlebox assigns external ports, the more random the strategy the harder and error-prone the prediction. However, the results of our field test conducted in chapter 4 show that approx. 57% (44% for TCP) use a rather straight-forward assignment strategy and allow predicting the external port by observing patterns. However, this is only true if cross-traffic from other hosts behind the same middlebox does not influence the port assignment strategy. The more active hosts behind one NAT the more unreliable a port prediction algorithm.

[63] describes the general requirements for port prediction (e.g. using Cone NATs that implement an independent binding) and also states that some Symmetric NATs assign port numbers incrementally, thus making port prediction possible. The authors mention timing issues when considering cross-traffic which leads to the impossibility of predicting ports for large installations. *NATBLASTER* proposes to use the birthday paradox for port prediction as already described above. [168] presents a port correlation algorithm for predicting external port numbers. Finally, [151] suggests to also take into account that other hosts in the same private network may also send out packets and may disturb the port prediction algorithm. Two methods are introduced: the capturing method captures all outgoing packets (including the ones from other hosts and applications) and helps to predict the influence of additional hosts. The scanning method actively scans the environment using tools such as ping and ARP. Based on neighboring nodes, an error rate for a predicted port can be given.

5.2.3 Additional Solutions

Data Relay

While establishing inbound connections is not always possible, outgoing connections to a peer located in the public Internet follow the intended packet flow and are therefore possible (if not blocked by a firewall). A data relay follows this approach by letting a client actively establish a connection and by forwarding incoming connections via the so-created socket. Traversal using Relay around NAT (TURN) [92] is the IETF approach for a data relay and developed as a STUN extension by using the same packet format and similar messages.

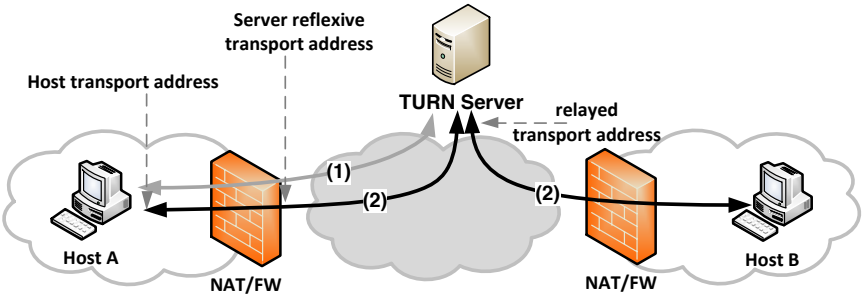

Fig. 5.9: Traversal using Relays around NAT (TURN)

Figure 5.9 depicts the details of TURN. In step (1) the TURN client requests a public endpoint from a TURN server containing the public endpoint of host B. The server receives the request and generates two endpoints: One endpoint that is used by the TURN client to communicate with the server and one "relayed transport address" that is used by host B for sending packets to host A. TURN has the advantage of working with almost every stateful middlebox since it is independent from the behavior of the middlebox and does not require explicit support of the middlebox implementation. However, the reliability and the throughput is dependent on the TURN server, which can be seen as a single point of failure.

Interactive Connectivity Establishment

The Interactive Connectivity Establishment (ICE) is "a protocol for Network Address Translator (NAT) traversal for UDP-based multimedia sessions established with the offer/answer model" [128]. ICE is able to utilize many traversal techniques and aims to select the one that works in the specific situation. Initially developed for VoIP, it is mostly used together with SIP and SDP [67] for enabling the traversal of media data.

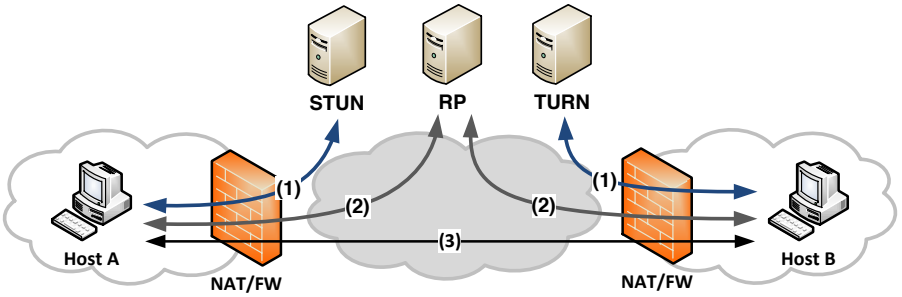

Fig. 5.10: The Interactive Connectivity Establishment (ICE)

Figure 5.10 shows the ICE approach. In step (1) hosts A and B collect possible mappings such as their local endpoints, as well as endpoints provided by STUN and TURN. These endpoints are exchanged in (2) via a rendezvous point, e.g. a SIP signaling infrastructure. Both hosts receive a list of remote endpoints, sort them in priority order and send checks for each pair using STUN messages. Once a working pair is detected it gets nominated for being used as a media endpoint. Dependent on the implementation the ICE algorithm either stops or searches for better and alternative pairs. An extension to ICE that supports TCP candidates also exists [129].

5.2.4 Evaluation and Comparison

As described in the last sections many proposals for middlebox traversal have been made. Each of them approach the problem in a slightly different way and each has advantages and disadvantages. While explicit solutions require active support of the middlebox, behavior-based solutions only work if certain requirements are met. Table 5.1 compares the discussed techniques and evaluates their properties.

The first property *performance* not only means the performance of the actual data connection, but also the overhead that is necessary to establish the according state in the middleboxes. This is indicated by using double (++) and (−−) signs for this category exclusively. For UPnP a short signaling period is needed only at the beginning of a connection, resulting in a single plus (+). For NSLP, MIDCOM, NUTSS and ICE

	UPnP	NSLP	MIDCOM	NUTSS	RSIP	SOCKS	ALG	STUN	HP	TURN	ICE
Performance	+	O	O	O	-	O	++	+	+	- -	O
Security	-	+	+	+	-	O	-	O	-	O	O
Supported Techniques	O	O	O	+	O	O	O	O	+	O	+
Flexibility	-	+	+	+	-	-	-	+	+	-	+
Support for legacy appl.	-	-	-	O	-	O	+	-	O	-	O
External Entities	+	-	-	-	+	+	+	-	O	-	-
Software on MB	-	-	-	-	-	-	-	+	+	+	+
Software on end-host	-	-	-	-	-	O	+	-	O	-	-

Tab. 5.1: State of the art summary

multiple messages or nodes are involved, resulting in a neutral rating (O) since the performance of the actual data connection is not affected. SOCKS is dependent on the proxy, RSIP tunnels packets to the gateway and TURN is completely dependent on the external TURN server that has to relay all traffic. UPnP, RSIP, ALG and HP do not provide *security* at all, STUN, SOCKS and TURN provide authentication and also a TLS mode and ICE's security mechanisms are dependent on the signaling protocol. MIDCOM uses PDPs, NSLP a complete authorization framework and NUTSS security is based on policy boxes and signaling, which results in a positive evaluation. Besides ICE, HP and NUTSS all other solutions are single *techniques* that either work (if implemented) or not. ICE and NUTSS allow utilizing arbitrary techniques and HP can be implemented in many different ways. Control-based frameworks provide a great *flexibility* and can be adapted to many situations, whereas ALGs are protocol specific and UPnP can either forward all packets to an internal port or block them. *Legacy Applications* are supported by ALGs if implemented and the NUTSS implementation proposes to use LD_PRELOAD in order to redirect socket calls to the NUTSS library. SOCKS is not only supported by many applications, but many operating systems also allow configuring a default proxy for all applications. UPnP, ALGs, RSIP and SOCKS are not dependent on *external entities* located in the public Internet. Hole punching might work without a third party in a few situations. Explicit solutions as described above need active support of the middlebox, if not implemented they don't work. Finally, only ALGs are completely transparent and do not require to install software on any end-host. Hole Punching might also work if the packets are sent in the correct sequence. SOCKS might work if the operating system allows setting a system wide proxy for all applications.

5.3 Application-Centric Middlebox Traversal

Each of the traversal techniques as discussed above has advantages and disadvantages and their success rate is either dependent on the behavior of the involved middleboxes (behavior-based solutions) or on the availability of external entities (explicit solutions). If both of these parameters are known a technique can be selected accordingly and applied to solve the traversal problem. The remaining question that has not been discussed in the state of the art is which technique is applicable for which application. We argue that a middlebox architecture can only become a valid design principle for the future Internet if the roles of applications and the intention of administrators are taken into account when traversing them. Thus, we defined four Middlebox Traversal Service Categories, each making assumptions about the purpose of the connection establishment (intention of the middlebox placement) and the infrastructure (e.g. external entities) that is available. Our categorization emphasizes that the applicability of many existing middlebox traversal techniques depends on the support of a combination of requester (client), the responder (service), globally reachable infrastructure nodes and the role of the application. On the one hand, server applications set up a socket and wait for connections (this also applies to P2P applications). On the other hand, client applications such as VoIP clients actively initiate a connection and wait for an answer on a different port (bundled session applications). Other applications only work across middleboxes if both ends participate in the connection establishment. Thus, we differentiate between supporting a service and supporting a client. Here, the client is called the requester, because it actively initiates a connection.

5.3.1 Service Categories for Middlebox Traversal

The first category **Requester Side Middlebox Traversal (RSMT)** covers applications that actively participate in the connection establishment and still suffer from the existence of NATs. This is true for the problem categories realm-specific addresses, for bundled session applications and for unsupported protocols. As depicted on the top left hand side of figure 5.11 middlebox traversal solutions that can be applied to the RSMT category need support of the requester side, either on the host or on the middlebox. Packet (1) is sent out towards the service in order to create a mapping. If this packet contains realm-specific addresses in the payload either the requester or the middlebox needs to translate them. The most popular application for RSMT is VoIP using SIP/SDP. For unsupported protocols a connection can only be established if the initial packet (1) is already prepared (e.g. encapsulated into UDP) in a way that is able to traverse the middlebox.

The second category, **Global Service Provisioning (GSP)**, covers applications and services that aim to be globally reachable. This service category applies to scenarios where middleboxes are deployed without the intention of breaking the end-to-end connectivity, e.g. NAT devices. To overcome this unwanted side-effect a peer hosting a service behind a middlebox needs to actively take care of establishing a mapping in a way that it can be accessed by arbitrary hosts located in remote networks (step 1 in figure 5.11). Some of the remote hosts may also reside behind stateful middleboxes and may not be able to participate in the traversal process. Globally accessible means that once a mapping has been created it will accept multiple connections from previously unknown clients. This is the main difference to RSMT, which only needs to create a mapping for one particular session (e.g. one call in case of VoIP). All legacy server applications (e.g. HTTP, FTP, IRC) are examples for the service category GSP. Addi-

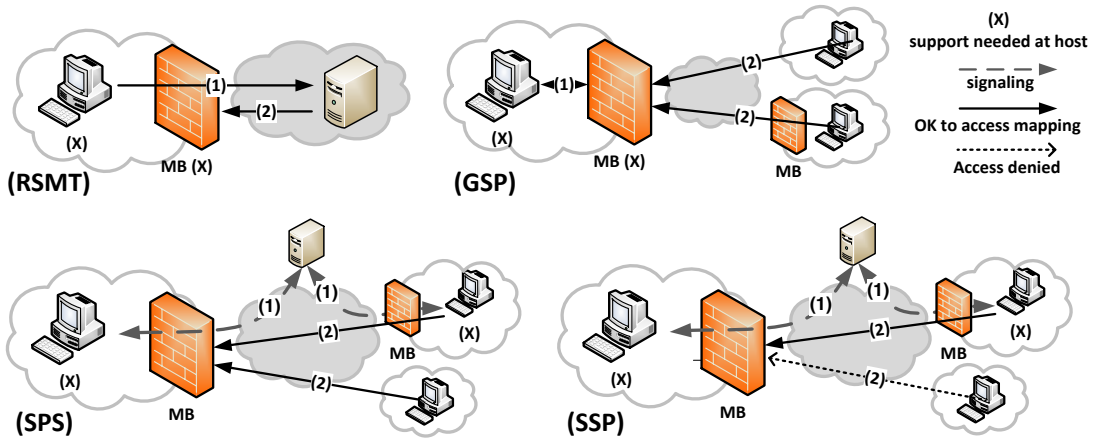

Fig. 5.11: Middlebox Traversal Service Categories

tionally, P2P applications that require incoming connections from previously unknown remote peers also fall into this category.

The third category **Service Provisioning with Signaling (SPS)** assumes support at both ends (the service and the requester) and does not make any assumptions about the accessibility of the mapping once it has been created. As shown in figure 5.11 this service category uses a signaling infrastructure (step 1) for exchanging connection specific information that helps to establish an appropriate mapping in the middlebox. SPS supports all kinds of services where a one-to-one connection is sufficient and signaling is available. Signaling also allows establishing a UDP tunnel between the hosts allowing unsupported protocols to traverse legacy middleboxes.

The fourth category, **Secure Service Provisioning (SSP)** addresses scenarios and applications that require certain restrictions for a mapping or rule created at the middlebox. This could be either an exclusive access of authorized hosts to a firewall or NAT mapping (as shown in figure 5.11), or a rather general restriction for using resources of a middlebox such as a proxy. This requires additional functionality for policy enforcement, which can be done at the middlebox itself, at a data relay or at a firewall. The example as shown in figure 5.11 uses signaling for authenticating a remote party before establishing a mapping, thus requiring support at both ends of the connection. However, it would also be possible not to use signaling by configuring rules or mappings for previously known clients or restrictions such as rate limits or bandwidth restrictions.

5.3.2 Application of Existing Traversal Techniques

Table 5.2 shows implications when applying state of the art techniques to our service categories. "Support at the S(ervice)/R(equester)/EX(tern)/MB" means that additional software needs to be deployed at this host. "Signaling messages" means that a signaling protocol is used for setting up a traversal method. A rendezvous point (RP) is needed for relaying signaling messages between the requester and service. Finally, "stream independent" describes the requirement for consecutive connections. For example, a port forwarding entry has to be created only once, while hole punching with restricted filtering requires to send a new hole punching packet for every new stream.

None of the existing techniques can be applied to all service categories. RSMT is required for applications and protocols that need active support to traverse a middle-

Serv. Cat.	Traversal Technique	Requires support at S	R	EX	MB	Signaling messages S	R	RP	EX	Stream indep.
RSMT	MB with ALG				•					
	UPnP		•		•		•			•
GSP	UPnP	•			•	•				•
	HP (indep. filt.)	•				•			•	•
	RSIP	•		•		•		•		•
SPS	HP	•	•			•	•	•	•	(•)
	UPnP	•	•		•	•	•	•		(•)
	data relay	•	•	•		•	•	•	•	(•)
	tunneling	•	•			•	•			•
SSP	HP (restricted filt.)	•	•			•	•	•	•	
	NSLP/MIDCOM	•	•		•	•	•			
	sec. relay (TURN)	•	•	•		•	•	•	•	
	secure tunneling	•	•			•	•			•

Tab. 5.2: Middlebox Traversal Service Categories and their implications for the SotA

box on outgoing packets. One common approach is therefore to integrate RSMT into theses applications (e.g. the VoIP client), allowing them to establish mappings on the fly. One possibility is the integration of a UPnP client. Another option is to use Application Layer Gateways (ALG) that interpret in-band signaling and establish mappings accordingly. ALGs are no general solution, because the middlebox must implement the necessary logic for each protocol and end-to-end security policies may prohibit the interpretation of the signaling messages by the middlebox.

GSP depends on traversal techniques that allow unrestricted access to a public endpoint. A control protocol can be used to directly establish a port forwarding entry in the mapping tables of the middlebox, for instance, with UPnP. Port forwarding entries created by UPnP are easy to maintain and work independently from middlebox behavior. However, UPnP only works for single middleboxes, since LSNs are (due to security reasons) not UPnP compatible. Hole punching is an alternative if UPnP is not applicable and works for independent filtering strategies. The mapping has to be refreshed periodically, for instance, by sending keep-alive packets. For middleboxes implementing a restricted filtering strategy, hole punching for GSP cannot be used since the source port of a potential requester is unknown in advance.

SPS makes no assumption about the accessibility of a created mapping, thus all possible techniques are applicable. Different to GSP, hole punching for SPS works as long as port prediction is possible. For middleboxes implementing restricted filtering, signaling helps to create the appropriate mapping since the 5-tuple of the connection is exchanged. Signaling also allows the establishment of a UDP tunnel, allowing the encapsulation of unsupported protocols. SPS can also use UPnP to establish port forwarding entries for one session.

SSP only allows authorized hosts to allocate and to use a mapping. Protocols that authorize requests and assume control over the middlebox, such as MIDCOM and NSLP qualify for SSP. UPnP is not useful for SSP, because it forwards inbound packets without considering the source transport address. Hole punching can only be applied if the middlebox implements a restricted filtering strategy. All cases discussed above rely

on additional measures to prohibit IP spoofing. The use of secure tunnels impedes IP spoofing and allows secure middlebox traversal, even for unsupported protocols (e.g. IPSec, SCTP, DCCP). SSP can also be achieved by using TURN with authentication, authorization and secure communication (e.g. via TLS).

Despite the great flexibility of SPS and SSP, both categories involve a number of assumptions that are not always satisfied. The most important one is the need for both ends (and sometimes also the infrastructure), to support compatible versions of the middlebox traversal framework. It remains to be seen if the future will bring a sufficiently large deployment of one framework to rely on for arbitrary applications. The chances are better within homogeneous problem domains, like telecommunication, where such frameworks can be integrated with the applications and be distributed in large numbers. For instance, the adoption of ICE is mainly happening within the VoIP/SIP community and focusing on VoIP specific use cases.

5.4 Summary and Key Findings

This chapter introduced the middlebox traversal problem and presented and evaluated state of the art middlebox traversal solutions. We argued that a successful middlebox traversal technique should not only enable the communication across middleboxes no matter what, but should also take the role of the application and the intention of the deployment into account. Thus, we defined four new service categories that consider the support of a combination of requester (client), the responder (service), globally reachable infrastructure nodes and the role of the application. This is the first step towards answering the question how to find suitable traversal techniques for applications (Q2 according to section 1.1). Finally, existing techniques were classified according to our service categories.

Key Findings of this chapter
(and contributions according to section 1.2):

- **Numerous state of the art traversal techniques exist and their success rate dependents on the behavior of the middlebox and the availability of external infrastructures.**

C4.1a **A middlebox traversal technique should not only enable the communication across middleboxes, but also consider authorization and accessibility requirements, as well as additional policies as defined by a user or administrator of a network.**

C4.1b **Middlebox Traversal Service Categories help to achieve this by taking the role of the application and available infrastructure into account.**

6. KNOWLEDGE-BASED MIDDLEBOX TRAVERSAL

6.1 Introduction

The traversal of middleboxes has been subject to many research projects as described in the last chapter. Besides individual approaches, frameworks (ICE) and applications (Skype) integrate many existing solutions to increase the probability for successfully establishing a connection through middleboxes. Since middlebox behavior is not standardized and has a large impact on the success rate of individual techniques, this approach seems reasonable. However, state of the art frameworks only aim at establishing a connection without considering the role of the application that can be expressed using our middlebox service categories. Solutions that also analyze the current situation, the behavior of the involved middleboxes and requirements of external entities such as administrators do not exist.

This chapter presents a new approach to middlebox traversal. The core contribution is an innovative framework that not only applies middlebox traversal techniques for the sake of connectivity, but also considers higher-level requirements of applications, users and administrators. To achieve this, elements derived from Autonomic Networking, such as an autonomic control loop, help to transform decisions into technical compromises and configurations. Such configurations are composed by considering and transforming instances of our information model that represents middlebox behavior, as well as policies introduced from external entities. Existing traversal algorithms and solutions can be easily integrated into the framework and are parameterized based on the exact policy for the session and the behavior of the involved middleboxes. This helps to support all of the four service categories as presented in the last chapter and allows applying the best applicable solution for the given scenario. We call this approach knowledge-based middlebox traversal because all decisions are made based on the knowledge about the network, the application and the participating entities. Besides being more flexible, the knowledge-based approach also has performance advantages over the state of the art due to decoupling the gathering of information from the actual traversal.

Section 6.2 introduces the general concept of knowledge-based middlebox traversal and shows the essential information that has to be gathered and maintained in order to make decisions. Section 6.3 describes a reference example for a framework implementing knowledge-based middlebox traversal. Section 6.4 then presents NOMADS, our framework that integrates existing and future traversal techniques and applies them according to policies and the behavior of the involved middleboxes. After giving an architectural overview we show how legacy as well as newly developed applications benefit from NOMADS. Existing traversal techniques are integrated, extended and parameterized according to the knowledge about the system and network. Our experimental results as described in chapter 4 are hereby the cornerstones for improving and adapting existing solutions. Additionally, two new middlebox traversal techniques are developed in section 6.5. After evaluating our approach in section 6.6, section 6.7 concludes this chapter.

6.2 Knowledge-based Middlebox Traversal

State of the art frameworks such as ICE query the network and determine working traversal techniques for each connection request separately, thus delaying the connection establishment. The idea of our knowledge-based approach, as initially published in [104], is to decouple the knowledge gathering process from the actual connection establishment. The knowledge about the network, involved middleboxes and applications can then be directly applied when a decision for a traversal technique is needed. The knowledge gathering process is iteratively ran in the background and not only makes sure that a working technique is determined, but also allows reacting to network and topology changes resulting in a fast connection establishment.

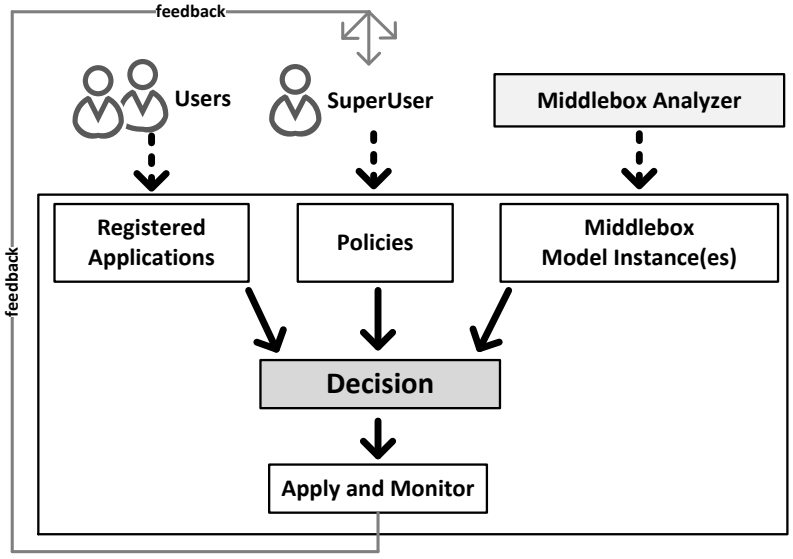

Fig. 6.1: Inputs to be considered for making decisions about suitable techniques

Figure 6.1 shows the components of our knowledge-based approach. Users register applications to the framework and define accessibility parameters for them. Each application may be assigned to a middlebox service category as described in chapter 5. This automatically limits the number of traversal techniques to be applied for the application, which is expressed by policies. Policies on the one hand are dynamically defined by privileged users to express authentication and authorization restrictions, but also hold rather static information about which middlebox traversal technique is applicable to which service category. Security policies will be covered in chapter 7 when introducing a new security architecture for unmanaged networks in general and for our framework in particular. For the application of middlebox traversal categories we maintain the following entries in the policy database:

(Service Category → Middlebox Traversal Technique → Condition)

For example, if an application is registered with the category Secure Service Provisioning (only authorized hosts are allowed to access it), the technique UPnP in general or hole punching in combination with an independent filtering strategy cannot be applied, because both violate the requirement of a secure public endpoint. However, a valid entry for SSP would list hole punching with SSP with the condition that the middlebox implements a restricted filtering policy. Conditions are linked to the instance of the

middlebox model as depicted on the upper right hand side of figure 6.1. The model instance is created by repeatedly running the measurement algorithms (integrated into the "Middlebox Analyzer") as defined and evaluated in chapter 4 and represents the exact behavior of the middlebox(es) towards the open Internet. It is important to note that although our topology analysis algorithm allows recognizing individual stateful middleboxes (or at least the number of hops towards the middleboxes), it is not possible to analyze all behavioral issues separately. Thus, if multiple middleboxes are found the created middlebox instance always represents the combined behavior, which is the desired behavior for establishing connections across the Internet. The Decision Module then takes the registered applications, policies and the middlebox model instance as an input and applies a middlebox traversal technique according to the situation. The outcome of the decision is constantly monitored and reported back using a feedback loop to users, as well as to the policies and measurement algorithms. This helps to be aware of the network and to react to changes accordingly.

6.3 Reference Examples for a knowledge-based Framework

Before actually presenting our new middlebox traversal framework in section 6.4 this section shows two reference examples to get an overall idea of our approach. This will help to better understand our design choices and the application flow in general. In the first scenario we assume an unilateral deployment where only one host runs an active instance and aims to make a service globally reachable. The second scenario then shows a complete example were both hosts run the framework and are thus able to coordinate on middlebox traversal through signaling.

6.3.1 Unilateral Deployment

Our first example assumes an unilateral deployment where the framework is only available at one host. Figure 6.2 shows a scenario where a user aims to make a service publicly available without knowing anything about a potential requester. In the first step the user communicates with the framework via a web-based interface (the Session Manager) and selects the web-server at port 80 from a list of running services. The framework instantly accesses its knowledge database and selects an appropriate middlebox traversal technique for the detected middlebox. Here, UPnP is selected and a public mapping is created. The framework also registers a dynamic DNS name for its external IP address and provides it back to the user. From now on all external services are able to connect to the DNS name to access the service.

6.3.2 Coordination of Traversal through Signaling

Figure 6.3 depicts a scenario where both ends run a framework instance and are able to use a public signaling infrastructure (here SIP) for exchanging connection specific information. A requester R aims to connect to a (web) service S running at port 80 using a TUN-base application interface that will be described in section 6.4. We assume that a privileged user of network B has registered the service at the framework and no policy exists that prevents the connection from being established.

First the requester R connects to *web.service.nat* which is resolved (by a DNS proxy) to S's SIP-URI (e.g. sip:service@sip.org). The application interface allocates an available IP address (here 172.16.1.200) from the range that is routed to the TUN device and reports it back as an answer for *web.service.nat*. This IP address only

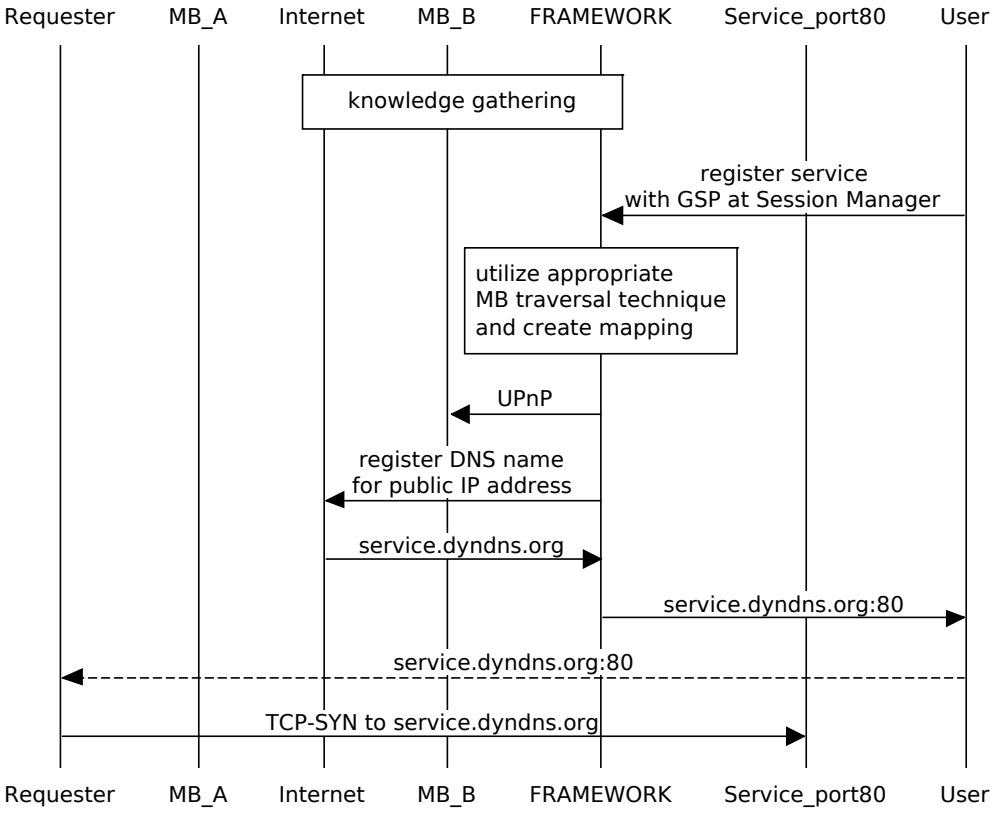

Fig. 6.2: Reference example for GSP with an unilateral deployment

needs to be unique on this host and does not represent the service in general (only the SIP-URI does). Host R now sends the first TCP-SYN to this address and a new mapping entry with a pending state is allocated. The packet is passed to the Decision Module of R, which decides to integrate information about the filtering element of the model instance into a so-called *Service Request*. The *Service Request* indicates that the requester aims to connect to the (local) destination port 80 and is sent via the signaling infrastructure to the SIP-URI. S looks up its Session Manager and finds an entry for port 80. From the *Service Request*, S knows that the requester implements a connection dependent binding strategy and assumes that R is not able to predict any external mapping. Instead of asking for a predicted port range (which would be a valid option), S selects hole punching as a valid technique for traversing the middleboxes as it implements an independent filtering strategy that forwards packets independent of the source address. S looks up the global mapping for port 80 and provides it to R (here 85.6.4.1:51543). R can now complete the mapping entry by adding this endpoint to its translation table and the held back initial TCP-SYN from the requesting application is translated and sent out to the real network interface. Since S's middlebox now has a mapping for this packet, it forwards it directly to the service. From now on every packet in both directions is rewritten according to R's mapping table. The following sections now present the individual modules in detail and help to further clarify the reference examples.

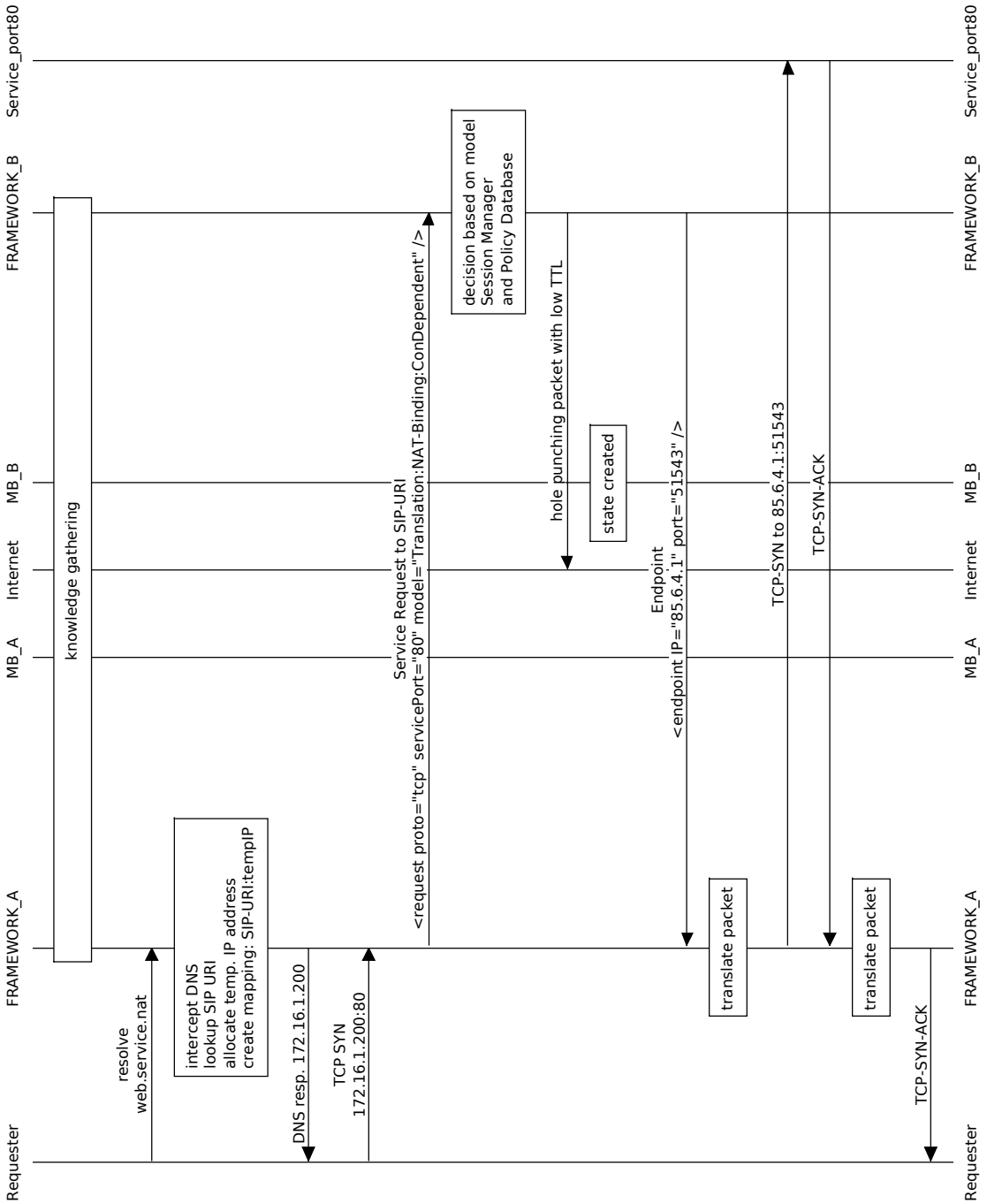

Fig. 6.3: Reference example for coordination through signaling

6.4 NOMADS: A new Middlebox Traversal Framework

The knowledge-based approach is the core element of our new k**NO**wledge-based **M**iddlebox tr**A**versal and **D**etection **S**ervice (NOMADS). The first instance of this framework was published under the name ANTS (Advanced NAT Traversal Framework) in [99] and [105] and has since been developed into a generic middlebox traversal solution integrating many existing and new techniques. One of the main requirements was not only to support different types of applications according to their service categories, but also to support legacy applications that do not have built-in middlebox traversal support. State of the art frameworks, such as ICE, have to be compiled into the application itself and therefore cannot support legacy applications. Furthermore, when establishing a connection between two hosts we cannot assume that both of them run the same framework. Solutions for coping with these scenarios have to be developed.

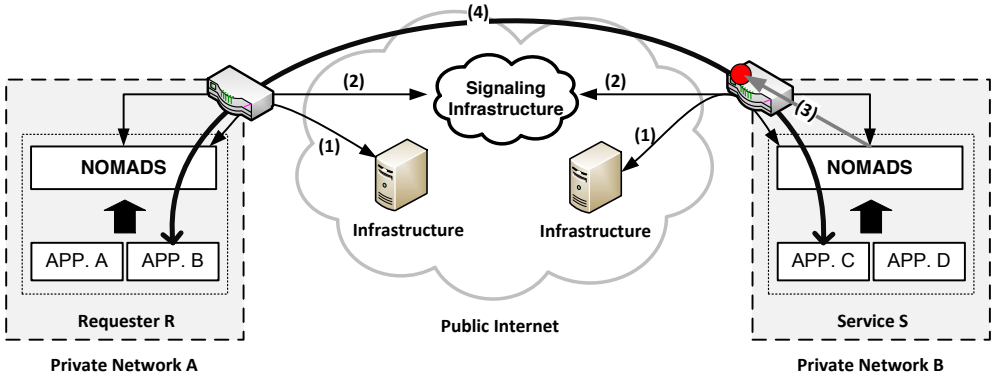

Fig. 6.4: Overview of NOMADS

Figure 6.4 shows an example of NOMADS where both hosts run an instance of the framework. NOMADS is only installed once on each host and supports all applications requiring middlebox traversal support. The connectivity establishment process depends on the available entities such as STUN servers, data relays and signaling infrastructure nodes. Here, application B at the requester R aims to connect to application C at the service S. First, NOMADS detects external infrastructures and gains knowledge about the network (step 1). On a connection request the NOMADS instances are paired using an external signaling infrastructure (2). After parameterizing a middlebox technique in step 3, the requester is finally able to connect to the public endpoint of S (step 4). This section presents our framework in detail. We first cover relevant scenarios and give an architectural overview. Afterwards, the individual modules of our architecture are described.

6.4.1 Scenarios

Our framework targets scenarios where middleboxes hinder a connection from being established. We do not differentiate between multiple layers of middleboxes or specific types of middleboxes. As defined above we aim to enable an application to communicate across middleboxes in a way that is compliant with authorization and accessibility requirements, as well as additional policies as defined by the user or the administrator of the network. In the following we present four example scenarios that are targeted by our framework. However, the applicability is not limited to the presented scenarios and each scenario could also deploy additional middleboxes that are not shown.

The first scenario covers **unmanaged networks**. The term "unmanaged networks" describes networks that are not professionally administrated. In many cases home networks are an example for unmanaged networks. The owner of the home receives a consumer router from his ISP (or has to buy it himself) and is responsible for configuring Internet access settings, IP connectivity for other devices and WLAN security parameters. The top row of figure 6.5 depicts two scenarios of unmanaged networks. On the left hand side both communication partners are behind a stateful home router and either connected to different ISPs (s1) or to the same ISP in scenario (s2). On the right hand side one of the ISPs also implements LSN. Applications, such as VoIP, that aim to establish a direct connection between two or more hosts located in unmanaged networks will benefit from our framework. Furthermore, applications that provide a service on a global endpoint, such as traditional server applications, are supported.

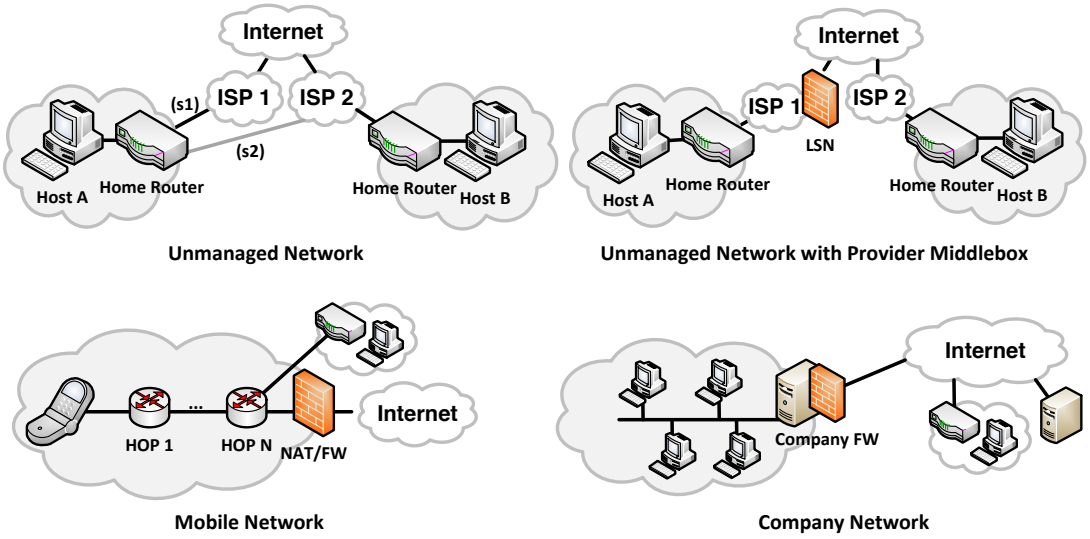

Fig. 6.5: Example scenarios targeted by our framework

Our second scenario, as depicted on the bottom left hand side of figure 6.5, targets **mobile users** that are affected by middleboxes deployed by their provider, such as LSNs, proxies and firewalls. Here we not only focus on smartphones, but also on unmanaged networks that use a mobile connection as their main internet service, for example via LTE in remote areas. In this case most of the unmanaged networks will also deploy a consumer router in their own network for sharing the IP address as provided by the ISP.

Finally we target **company networks** from small unadministrated start-ups to larger networks. In such networks the administrator usually deploys a firewall to block and filter traffic based on the companies policy. On the other hand many modern protocols (mostly bundled session applications) fail when using static rules and scenarios where hosts are temporarily authorized to provide a globally reachable service are not possible.

One essential requirement for our middlebox traversal framework is that we don't require both ends of a communication path to install additional software. Thus, NOMADS can either be installed on one end-host, on two end-hosts to allow signaling, or on middleboxes only. Dependent on the deployment, more or less middlebox traversal techniques are applicable. For example, a connection between two middleboxes both implementing port address restricted filtering cannot be established when using hole

punching without signaling. Different to state of the art frameworks, our framework is only installed once per host and is able to support all applications that are registered to it. This is a huge advantage since it allows different applications to cooperate and to share their knowledge about existing connections and about the success rate of applied traversal techniques.

6.4.2 Architectural Overview

The modular architecture is depicted in figure 6.6 and consists of three layers and six individual modules. The layers can be seen as logical layers, where the top layer implements the registry point for applications, the policies and the actual middlebox model instance as described in section 6.2. The middle layer implements the Decision Module. The lower layer modules are used to communicate with the network.

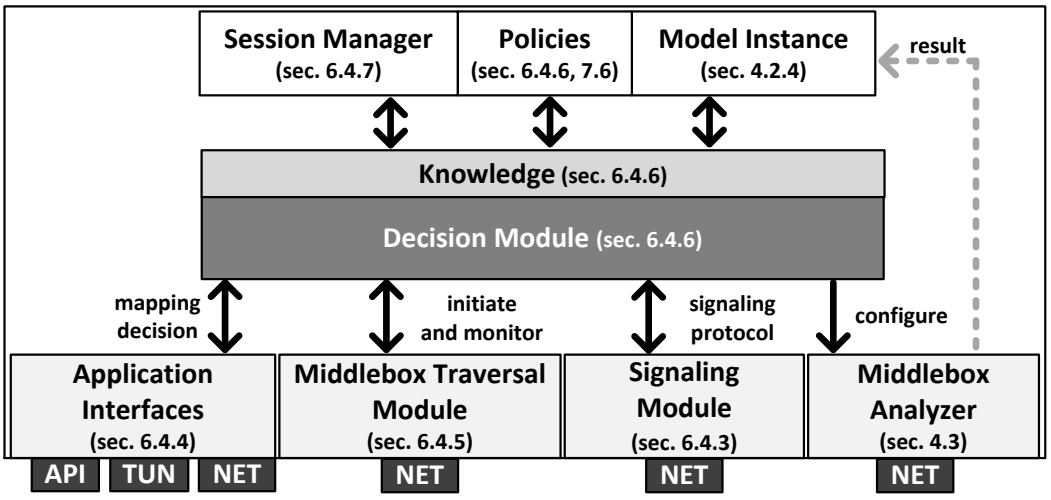

Fig. 6.6: Modular architecture of NOMADS

Whenever an application needs middlebox traversal support, a privileged user registers it at the Session Manager and assigns a service category to it (e.g. a web server needs to be globally reachable, thus GSP). The framework is then able to establish connectivity using one of its middlebox traversal techniques located at the Middlebox Traversal Module. The selection of a method depends on many factors such as the behavior of the middlebox, potential requesters (not all of them also run an instance of the framework) and the role of the application. As described above, the Middlebox Analyzer periodically updates the middlebox model instance and together with the Session Manager and the policies, knowledge about the network is created. Whenever a decision is needed, the Decision Module reuses this knowledge in order to parameterize an appropriate middlebox traversal technique. If an application is registered at the Session Manager with the service category GSP and an "all allow" policy, the Decision Module immediately invokes a middlebox traversal technique suitable for this service category, e.g. UPnP. Additionally, a dynamic DNS name is assigned to the public IP address and reported back to the user who is then able to distribute the so-created public endpoint to potential external users.

Other connectivity scenarios (SPS and SSP) require a coordination of NOMADS instances on the requester and service. In this case two other modules are needed: the Signaling Module on both hosts for exchanging connection specific information

(described in section 6.4.3), and the Application Interface Module at the requester for connecting applications to the created public endpoint as described in section 6.4.4.

6.4.3 Signaling Module: Request Response Protocol

If both hosts of a connection run the NOMADS framework the Signaling Module can be used to exchange information about an upcoming connection. This allows perfectly adapting and parameterizing traversal techniques based on the involved middleboxes and the current behavior of the network. Furthermore, signaling allows the service to only allocate a mapping if the requester is authorized to access it. More generally, signaling first informs the service about the source of the connection, while in the second step the service reports the created endpoint back to the requester that is then able to use it. Instead of exchanging a number of candidates and finding out which one actually works (ICE), NOMADS uses its knowledge about the middlebox(es), the application and the requester to allocate a working endpoint resulting in a fast connection establishment.

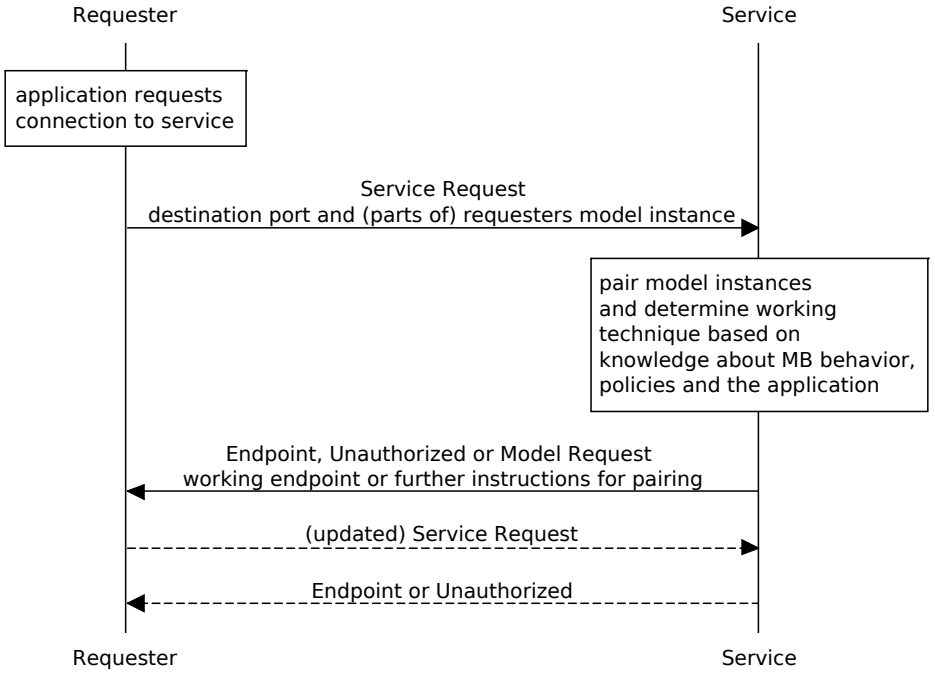

Fig. 6.7: Agreeing on a method using signaling

In NOMADS each host has a unique URI (e.g. a SIP-URI [131]) and is registered at a public rendezvous point (e.g. a SIP proxy). The URI can be seen as an approach for splitting the identifier and locator of a host as also proposed by HIP [98] and NUTSS [61]. The signaling protocol is based on XML and can be easily used with an arbitrary signaling infrastructure, such as SIP and XMPP (Jabber) [135]. We propose to use SIP messages for signaling, including the decentralized P2P-SIP[1] approach. Whenever a peer wants to establish a connection to another peer it sends the signaling messages through the signaling infrastructure to the host that runs a service. An URI identifies a host behind a middlebox according to its layer 3 address and multiplexing between different applications is done via the service port that is included in the so-called *Service*

[1] http://www.p2psip.org/

Request (see figure 6.7). The *Service Request* asks for a specific service (port) on this host. The service is then responsible for allocating an appropriate endpoint according to local policies and sends it back to the requester (via the *Endpoint* message). The main advantage of our signaling process is that for basic signaling (service category SPS) only two messages are needed. For the secure version, Secure Service Provisioning (SSP), we rely on authentication mechanisms as provided by SIP: The well known Digest Access Authentication [51] increases the number of signaling messages to four. A *Service Request* is answered with an *Unauthorized* messages and a second *Service Request* containing the computed response is sent. However, if a PKI exists (as described in section 7) we are able to use S/MIME [122] as defined in the original SIP RFC. The tunneled "message/sip" mechanism can be avoided by using a SIP Authenticated Identity Body (AIB) [116]. If the service feels that it needs more information from the requester after a *Service Request*, it may send a *Model Request* asking for specific middlebox behavior issues. The requester will then decide again which information it is willing to provide and reply with another *Service Request*. The complete signaling protocol is shown in Appendix C.2.

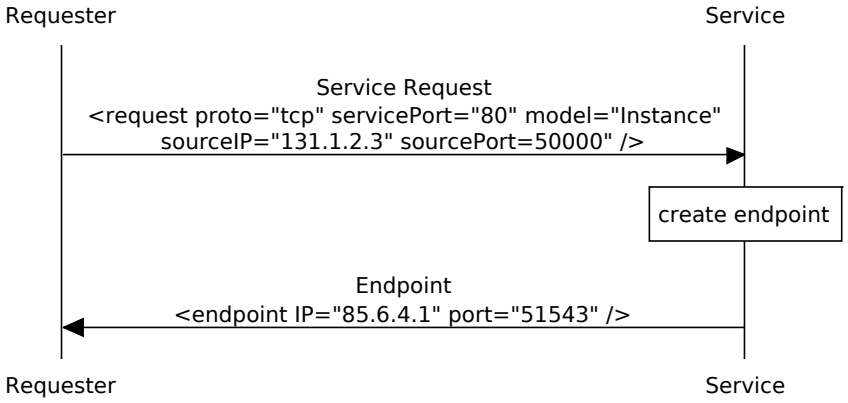

Fig. 6.8: Basic signaling messages for SPS

Figure 6.8 shows a more detailed example of the signaling protocol. The requester on the left aims to access an application that is running on the local port 80 at the service. Please note that NOMADS always works with local service ports, the actual translation and mapping is completely transparent to the user, which is an enormous advantage regarding usability. Thus, the requester sends a *Service Request* containing the destined service port 80 and the protocol TCP. The model instance is either complete or in parts provided in the model field (depending on the requester's policy) and the requester may also provide information about the source of the upcoming connection. In the example showed in figure 6.8 the requester was able to predict the external mapping of the actual (upcoming) connection, thus helping the service to take care of either accessibility policies or to initiate techniques such as hole punching. After receiving the *Service Request* and after creating a mapping the service sends the public endpoint *85.6.4.1 port 51543* back to the requester. In this example the service's middlebox implements non-port preservation, resulting in a "random" port 51534 that is forwarded to the local port 80. To reach our goal of transparency the next section shows how to map between the local port 80 and the provided external port.

6.4.4 Application Interfaces

When connecting applications that utilize the signaling infrastructure to NOMADS there are two options: the NOMADS socket API as described in Appendix C.1 allows newly created applications to easily use the framework. The requester application issues the *connect* function with the URI identifying the service as the destination. NOMADS then opens a real network socket to connect to the public endpoint created after exchanging information through signaling and finally translates between the sockets. Since legacy applications are not linked against the API yet, we also implemented a TUN-based solution. A TUN device is a virtual network interface that delivers packets to the user-space of a program and is well known from applications such as OpenVPN[2]. With this approach the requester application only needs to send the packets to the TUN device which then forwards them to NOMADS. NOMADS initiates signaling based on these packets and translates them to the allocated public endpoint at the service. From now on, NOMADS maps all packets coming from the TUN device to the public endpoint and vice versa.

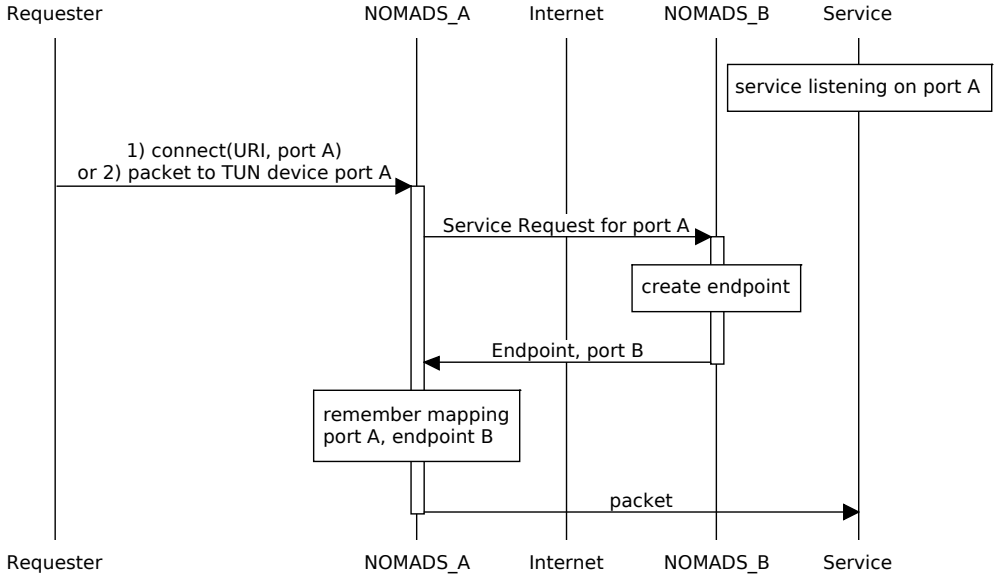

Fig. 6.9: Connecting applications to NOMADS

Figure 6.9 shows this process in detail. In order to trigger the NOMADS framework the requester either uses the *connect* function of the socket API or sends a new packet to the TUN interface. As described above, hosts are identified via a URI, thus the URI is directly provided to the *connect* function. When using the TUN interface a temporary IP address is used to communicate between the application and the NOMADS framework. This can be achieved by using a DNS service that returns a temporary IP address for the service's DNS name as described in section 6.3. In either case the requester uses the local port of the service (here port A) for sending packets. The initial packet is intercepted at the NOMADS instance and a *Service Request* is sent. After receiving a working public endpoint, NOMADS establishes a connection to it. In case of TUN, NOMADS also has to translate between the local (temporary IP address and port A) and the public endpoint (as provided by NOMADS_B). Here, the functionality is very similar to a traditional NAT44.

[2] http://openvpn.net/

6.4.5 Integration and Adaption of Middlebox Traversal Techniques

The Middlebox Traversal Module integrates actual middlebox traversal techniques and provides interfaces for the Decision Module to parameterize them. The primary goal is to use the knowledge about the middlebox instances for selecting and setting up the most appropriate technique for the situation. This section shows the integration, adaption and extension of existing traversal techniques. In the following section we then present the Decision Module.

Explicit Solutions

Explicit middlebox traversal solutions as presented in section 5.2.1 can be integrated into the Middlebox Traversal Module by using the following interfaces:

Technique:	UPnP | NAT-PMP | MIDCOM | NSIS/NSLP | SOCKS
Protocol:	transport layer protocol (TCP or UDP)
Destination:	local endpoint for redirection
Source:	optional: the source of the actual connection
Security Token:	optional: authentication token

Dependent on the actual technique, different parameters are necessary. Since UPnP cannot limit the accessibility of the mapping to selected hosts only, a wildcard is used for the source parameter. Other techniques such as MIDCOM, NSIS/NSLP and SOCKS also support authentication and need a security token for configuring the actual mapping.

Behavior-based: Hole Punching

The Middlebox Traversal Module integrates different behavior-based versions of hole punching for UDP and TCP as described in sections 4.6.5 and 5.2.2. As a result of our field test we stated that "a single traversal mechanism can never be as effective as a solution that carefully parameterizes a basic traversal algorithm according to the current situation, the topology and the behavior of the involved middleboxes". Especially different settings of the TTL value for the actual hole punching packet increases the probability of a successful connection. In case of a home router that is usually deployed only one hop from the end-host, the TTL value can be set to 2. This causes a packet to expire in the ISPs network and prevents it from reaching the remote middlebox. However, if middleboxes are not only deployed at the edges of a network, but also within the network itself (e.g. LSN at the provider) a static TTL value of 2 is not sufficient. The question here is how to detect an appropriate TTL value that still traverses all outgoing middleboxes, but expires before it reaches the outermost middlebox of the remote host. In the state of the art [151] proposes to first run the traceroute algorithm to detect the number of hops between two hosts and set the TTL value to half of it. With our topology detection algorithm as presented in section 4.3.4 we are able to determine the exact position (and sometimes even the behavior) of involved middleboxes and therefore set the TTL value dynamically (as described in section 6.4.6). For the Middlebox Traversal Module it is only important to offer an interface for setting the TTL value dynamically. Thus, the interface for hole punching traversal techniques is as follows:

Protocol:	transport layer protocol (TCP or UDP)
Source:	source address of the hole punching packet
Destination:	destination address for the hole punching packet (may be a port range for error-prone mappings)
TTL:	TTL value for the hole punching packet

Dependent on the actual technique that is chosen by the Decision Module, hole punching is instantiated by providing the protocol, the source address, the destination address and the TTL value. For example, if an application listening on port 20000 needs middlebox traversal support, the hole punching packet needs to be sent from this source port in order to create the appropriate mapping in the stateful middlebox. The destination address is the source address of the actual packet that is sent by the requester to initiate the connection and provided within the *Service Request*.

Behavior-based: Endpoint Prediction

Endpoint prediction as described in section 5.2.2 is necessary for behavior-based traversal such as hole punching. For endpoint independent binding strategies the STUN protocol can be used to ask an external service for the current public endpoint of a connection and reuse it for the actual data connection. For connection dependent binding strategies STUN is not applicable. However, our field test showed that port prediction is still possible with many connection dependent bindings by analyzing port allocation patterns. Furthermore, analyzing a sequence of n external mappings often also allows predicting the $n + 1$ mapping dependent on the mapping category as presented in section 4.6.3. The port prediction part of the Middlebox Traversal Module provides the following generic interface:

Input:	private (local) endpoint
Output:	predicted external endpoint or range of endpoints plus error estimation
Model:	optional: part of the middlebox model instance

The Decision Module provides a private endpoint to the prediction module and expects a public one in return. The returned public endpoint may be precise or it may consist of a port range plus an error estimation as described in section 4.6.3. As an option, parts of the middlebox model instance may also be provided to allow the prediction of ports based on the behavior of the middlebox. Cross-traffic can also be considered by the use of techniques as presented in section 5.2.2. By combining these techniques and considering the findings of our field test, port prediction with NOMADS is more efficient that in the state of the art.

Data Relay

As a last resort data relays such as TURN are a valid choice for establishing a connection between two hosts. The client part of the data relay is integrated into the Middlebox Traversal Module and the source, destination (optional) and data relay to be used has to be specified. Additionally, in [69] we extended the TURN protocol in a way that it also supports our middlebox traversal service categories. The key contribution hereby was to make sure to allow arbitrary sources, but at the same time not to overload the data relay. This was done by a metric that monitors the load and used bandwidth of the relay to regulate the access to mappings. The extensions were integrated into an existing TURN implementation and are fully compliant with the existing protocol.

6.4.6 Decision Module

The Decision Module together with the knowledge instance is the core component of our framework and acts as the interface between the Input Module (containing policies, model instances and registered applications) and the underlying modules (Application Interface, Middlebox Traversal Module, Signaling Module and Middlebox Analyzer) as described above. The Decision Module also monitors its decisions, learns from them and provides feedback to the user, as well as to the actual technical modules. The knowledge instance can be seen as a dynamic database that constantly gathers data from the individual modules, aggregates and stores them for further processing. Figure 6.10 shows the internal steps of the Decision Module. Once it is triggered by a *Service Request* or by the Session Manager it uses the knowledge about the involved middleboxes (which is expressed by the model instances) for determining a list of working techniques (Pairing). The final decision is then made based on this list and additional policies. Finally, the traversal is applied and feedback about the applicability and success rate is provided. In the following sections we first describe the general approach of the Decision Model, when it is conducted and which individual decisions (in addition to the selection of a working technique) have to be made. Afterwards the pairing process and the policies are presented.

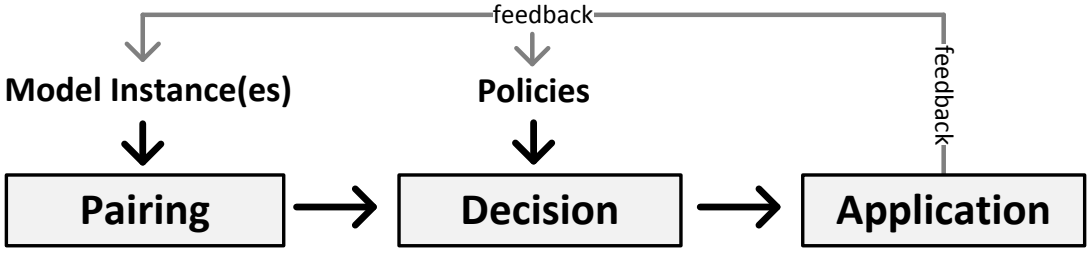

Fig. 6.10: Internal steps of the Decision Module

General Approach

The Decision Module is conducted on the following events:

- An application is registered to the Session Manager and requests a globally reachable endpoint (category GSP): The Decision Module notices the new registration, checks with local (security) policies, queries the model instance and initiates a middlebox traversal technique suitable for the application. The so-created endpoint is then reported back to the user. A complete example is given in section 6.3.1.
- A local application initiates a connection: the connection is registered at the Application Interface (NOMADS socket or TUN) and the Decision Module is asked to initiate the request response protocol via the Signaling Module. In this case the NOMADS instance acts as the requester of a session and decides which parameters should be passed to the remote host via the *Service Request*.
- A remote application initiates a connection: in this case the host that runs the NOMADS instance hosts a service and will be responsible for invoking an appropriate middlebox traversal technique. Once the Signaling Module receives a *Service Request* it passes the containing information to the Decision Module, which then decides how to process it. First, the Session Manager and the lo-

cal policy database is looked up to check if the application was registered and which service category is assigned to it. This knowledge together with the local middlebox model instance is then used to invoke a middlebox traversal technique and to create the response to the *Service Request*. An example of two NOMADS instances that use signaling for coordination is given in section 6.3.2. In addition to invoking a middlebox traversal technique, the Decision Module also has to take care of keeping mappings alive, e.g. by monitoring the *Stateful:StateTimer* and by sending keep-alive packets.

- An environment change is detected: the environment (e.g. network links) is constantly monitored and if a change is detected the Decision Module invokes the Middlebox Analyzer to gather knowledge about the new environment. Once the knowledge gathering process is completed the Decision Module compares the most current middlebox model instance with the previous one and has to react accordingly, (e.g. by reconfiguring mappings).
- A policy is updated: the Decision Module monitors the Session Manager as well as the local policy database and has to react to updates as provided by the users. For example, if the service category of an application changes from unrestricted (GSP, SPS) to Secure Service Provisioning (SSP), the Decision Module has to take care of "closing" existing connections and invoking new ones by using techniques that are compliant with the service category.

Pairing

The most important task of the Decision Module is to find a working technique based on the behavior of involved middlebox(es). More precisely, values to pass to the interfaces for the individual middlebox traversal techniques as presented in section 6.4.5 are determined. Again, we have to differentiate between two scenarios. For the service categories RSMT and GSP only one host runs an instance of the framework and chooses a technique based on its own middlebox behavior. For the categories SPS and SSP the service is able to pair the requester's model with its own to determine working techniques. However, only parts of the requester's model may be provided to the service, dependent on the policy as described in the following section. The actual pairing is usually done at the service side that has to invoke the middlebox traversal technique, but if we also want to allow the "swap role" functionality (as proposed in section 4.6.2), the requester side may be chosen to become the service. Thus, the pairing component expects the model instance of the service plus (optionally) additional information of the requester in order to generate a list of working techniques as its output. We now look at the individual techniques and list the prerequisites and rules for generating the so-called "candidate list".

Explicit middlebox traversal techniques (see sections 5.2.1 and 6.4.5) are not dependent on the behavior of the involved middleboxes. If the technique is supported by all middleboxes along the path from the open Internet to the service, it can be applied. The Middlebox Analyzer tests for their support and their availability is reflected in the *ProtocolLayers* element. For example, if a UPnP enabled Internet Gateway Device is detected the model instance contains the following entry: *ProtocolLayers:7:UPnP*. For some techniques it is essential to know the source of a connection in advance, thus they are only applicable if the requester also supplies its predicted source address. Finally, for explicit techniques we have to make sure that all hosts on the path support the explicit technique. Therefore, the Middlebox Analyzer also runs a topology test to detect the number of middleboxes on the path towards the open Internet. This knowledge

is then concatenated with the availability of the individual technique. For example, if UPnP is available in the model instance it can only be applied if the topology test found exactly one stateful middlebox towards the Internet.

As the name states, the applicability of behavior-based techniques depends on the behavior of the involved middleboxes. Hole punching is the key technique that enables a direct connectivity between two hosts across multiple layers of middleboxes and is also the prerequisite for the tunneling of unsupported protocols. There are a few things to consider before deciding if hole punching works and which parameters should be used as an input for the hole punching module. First, the predictability of the external mapping at the server side is needed. If the service is not able to predict the mapping (and pass it to the requester) the requester cannot connect to it. As described in section 6.4.5 the endpoint prediction part of the Middlebox Traversal Module returns a predicted public endpoint for a private one. If port prediction at the service side is not possible, but the requester implements an independent filtering strategy (*Filtering:State-based:Independent*) and supports port prediction, swapping roles is still possible. If an error-prone endpoint is returned, hole punching can still be tried using multiple connections simultaneously. Endpoint prediction on the requester side is also necessary for hole punching with restricted filtering on the service side. As described above, the requester queries the port prediction part of the Middlebox Traversal Module and includes the result in the *Service Request*. Again, for error-prone mappings a port range can be provided and multiple simultaneous connections increase the chance of a successful connection.

Secondly, the service needs to prevent the middlebox from closing the mapping by accident. For example, when sending out a UDP hole punching packet towards the requester, an ICMP port unreachable messages may be sent as a reply, which may close the mapping before it can be accessed by the actual connection. The same is true for TCP, where ICMP port unreachable messages or even TCP-RST packets are possible. For more details on hole punching please see section 4.3.2. This behavior is listed in the *Service:Stateful:StateRemovePolicy* element and the service has to consider the following: it first has to check its own behavior for packet sequences that close the connection. This has to be paired with the behavior of the requester *Requester:Stateful:NoStatePolicy* to finally determine the TTL value of the hole punching packet. For example, if the service's middlebox closes a mapping after seeing a TCP-SYN/TCP-RST sequence and the requester's middlebox actually sends a TCP-RST when receiving a TCP-SYN on a non-existent mapping the service sets the TTL field to a value smaller than the number of hops to the requester's middlebox. Table 6.1 shows possible middlebox combinations and the implications for hole punching. Here we assume that endpoint prediction is possible for both hosts. In Appendix C.1 we list a complete example of the decision tree for hole punching. After the pairing process the Decision Module has determined a list of working techniques that is, according to figure 6.10, passed to the actual decision part for further processing and applied afterwards.

Policies for Decision

After a successful pairing process an ordered list of working techniques exists (candidate list). Before choosing the actual technique to be applied and parameterized further, the Policy Module is conducted to make sure the selected technique is compliant with the systems policies. Thus, the Policy Module holds rules that influence the behavior of the framework. Some of the rules are rather static, others are dynamically updated by the feedback loop of our system to react to network changes and to learn from former

Requester	Service	Implication
(∗)	*Filtering:State-based:Indep.*	plain HP
Filtering:State-based:Indep.	(∗)	swap roles
Stateful:NoStatePolicy: UDP in, dest. unreachable out	*Stateful:StateRemovePolicy*: Remove on dest. unreachable	set low TTL
Stateful:NoStatePolicy: TCP-SYN in, TCP-RST out	*Stateful:StateRemovePolicy*: Remove on TCP-RST	set low TTL
(∗)	*Stateful:StateRemovePolicy*: Remove on ICMP TTL exceeded	set high TTL

Tab. 6.1: Possible model instance combinations and their impact for hole punching

decisions. The following listing gives an overview of the policies that are maintained:

- System policies for setting up the framework.
- Security policies defining who is allowed to access which service.
- Privacy and disclosure policies for requesters.
- Policies about the applicability of traversal techniques to service categories.
- Policies about the applicability of traversal techniques to applications.

The very basic configuration of NOMADS consists of an URI that is configured by privileged users or administrators. The URI serves as the globally unique NOMADS identifier (as described above) and is used by other hosts to contact the framework before initiating middlebox traversal. Additional system parameters are DNS names and information about available external infrastructure.

The Policy Module also holds security policies to express who is allowed to access which service in case of Secure Service Provisioning. As described in section 6.4.3 NOMADS supports digest access authentication with text-based user-names and passwords. However, for the S/MIME approach, digital identities (certificates) are required. Since digital certificates and public private key infrastructures (PKIs) are complex to maintain and not suitable for unmanaged networks, chapter 7 presents a secure service infrastructure that allows issuing digital certificates to entities in a user-friendly way, thus providing secure identities for NOMADS. As the technical structure of our security policies, the authorization framework XACML [97] is used (see chapter 7).

As part of the security (and privacy) policies, NOMADS defines which parts of the middlebox model instance are sent to the remote party in case of an outgoing *Service Request*. The more information NOMADS provides to the remote party, the better the customized parameterization of the actual middlebox traversal technique. For example, if the requester provides a predicted port within the *Service Request* the service is able to initiate hole punching and report a working endpoint back to the requester. If no information is provided, the service can only use hole punching if it implements an independent filtering strategy. Thus, as the default policy, the Decision Module of the requester invokes the port prediction part of the Middlebox Traversal Module and includes the predicted external endpoint to allow hole punching if possible. Additionally, information about the filtering element and typical packet sequences of the information model are also provided.

When describing the general knowledge-based approach to middlebox traversal in section 6.2 we also mentioned that a condition should be assigned to each traversal technique that has to be true before applying it to a service category. Table 6.2 lists the rather static mappings and conditions that are mostly dependent on the actual middlebox model instance. Therefore, we also refer to the model elements as presented in section 4.2. Whenever a new middlebox traversal technique is added to the framework, a corresponding policy has to be set up.

Service	**Technique**	**Condition**
RSMT	All techniques	none
GSP	UPnP	none
GSP	Hole Punching	*Filtering:State-based:Independent*
GSP	RSIP	none
GSP	SOCKS	none
GSP	Data Relay	*Relay:Filtering:State-based:Independent*
SPS	All techniques	none
SSP	Hole Punching	*Filtering:State-based:Restricted*
SSP	Data Relay	*Relay:Filtering:State-based:Restricted*
SSP	Tunneling	Secure Tunnel Endpoint
SSP	NSLP/MIDCOM	Authentication

Tab. 6.2: Policies for the applicability of middlebox traversal techniques to service categories

The last policy set that is held in the Policy Module defines which technique should be used for which application. Dependent on the middlebox behavior each technique may have to be applied only once to establish a connection, or it has to be applied many times if an application uses multiple simultaneous connections (stream independency according to section 5.3). For example, a translating middlebox implementing restricted filtering will have to utilize hole punching for every new IP5 tuple. Protocols such as http use a different source port for each http request, thus resulting in many hole punching packets for accessing only one website. UPnP on the other hand only opens a port once and accepts packets from arbitrary sources, just like independent filtering. The stream independency column as shown in table 5.2 (section 5.3) gives an overview of the behavior of the individual techniques. The applicability policy (implemented as a priority list) can be set manually, but is also continuously updated as part of the feedback loop of the monitoring functionality.

Application

Finally, the application part of the Decision Module (see figure 6.10) receives the selected traversal technique and its parameters and invokes the appropriate function in the Middlebox Traversal Module. The application part also monitors the result of the applied technique and gives feedback to the pairing and policy part of the Decision Module. This is an important step to always be aware of changing environments and to learn from previous decisions.

6.4.7 Implementation

The NOMADS framework was implemented as a proof of concept for our approach using the programming language C and Linux as an operating system. The algorithms for analyzing middlebox behavior as presented in chapter 4 were extracted from the field test software and included in the Middlebox Analyzer module that produces XML instances of the involved middleboxes. For the Signaling Module we utilize the eXtended osip library eXosip[3] and send the messages of the request response protocol (Appendix C.2) as XML payload embedded in SIP messages. This allows easily exchanging the underlying signaling infrastructure. The algorithms in the Middlebox Traversal Module were also extracted from the field test and adapted to match our interfaces as described in section 6.4.5. For the Application Interface Module we implemented the socket API as listed in the Appendix C.1, as well as the TUN-based approach using virtual network interfaces in Linux. The Decision Module can be fed with text-based policies in order to create a candidate list through pairing and to eventually decide which middlebox traversal technique should be utilized. Finally, the Session Manager was implemented as a web-based tool that runs on OpenWRT[4]. This allows runing only one centralized Session Manager in a network (e.g. on the home router) that communicates with the individual NOMADS instances running at the hosts. The prototype implementation is also used for the evaluation of NOMADS.

6.5 New Middlebox Traversal Techniques

As presented in our state of the art survey in section 5.2, there are many existing solutions for middlebox traversal. When evaluating existing techniques in the context of measuring middlebox behavior in chapter 4, we also gave recommendations on how to improve these techniques. The parameterization was then done in section 6.4.5. This section presents two new approaches for middlebox traversal that not only improve existing techniques, but explicitly solve two open problems: in section 6.5.1 we propose a DPWS-enabled Internet Gateway Device that enables a UPnP-like service by also providing security and in section 6.5.2 an approach for hole punching without the need of a third party is presented.

6.5.1 Devices Profiles for Web Services IGD

UPnP is a useful and widely deployed technique that allows automatically configuring port forwarding entries and can be seen as a good choice for the service category Global Service Provisioning (GSP). However, due to its insecure architecture that offers neither authentication nor a secure communication channel, router manufacturers often disable UPnP functionality by default. With UPnP, any host in the network is able to discover the IGD service and establish, modify or delete a port forwarding entry. An attacker may manipulate existing connections or malware may try to forward a port on the router in order to allow an attacker to access a host located in the private network. Some UPnP devices are even vulnerable to attacks initiated from outside the private network. As shown in [113], a cross-site scripting attack using a *XMLHttpRequest* object can be used to execute a custom HTTP request, e.g. the SOAP action “AddPortMapping” to open a port on the router.

[3] http://savannah.nongnu.org/projects/exosip/
[4] http://www.openwrt.org

To target these problems we developed an Internet Gateway Device (IGD) on top of the Devices Profile for Web Services (DPWS) [31], an architecture offering service discovery, description and eventing in a secure and standardized manner. The following sections shortly introduce DPWS, show the design goals of our IGD and describe the architecture and implementation.

Devices Profile for Webservices

DPWS defines a "minimal set of implementation constraints to enable secure Web Service messaging, discovery, description, and eventing on resource-constrained endpoints" [42] and was originally developed by Microsoft as a possible successor for UPnP. DPWS terminology distinguishes between two types of services: a hosting service representing a device (or a server) and a hosted service, which is the actual service as provided by the device. Hereby, a hosting service can host multiple hosted services. DPWS not only builds upon web services standards such as the Web Services Description Language (WSDL) [23], XML Schema [149], SOAP [57], WS-Addressing [58], WS-Eventing [93] and WS-Discovery [96], it also supports a subset of the security features as defined in WS-Security [107]. This allows signing multicast discovery messages and encrypting the actual data exchanged.

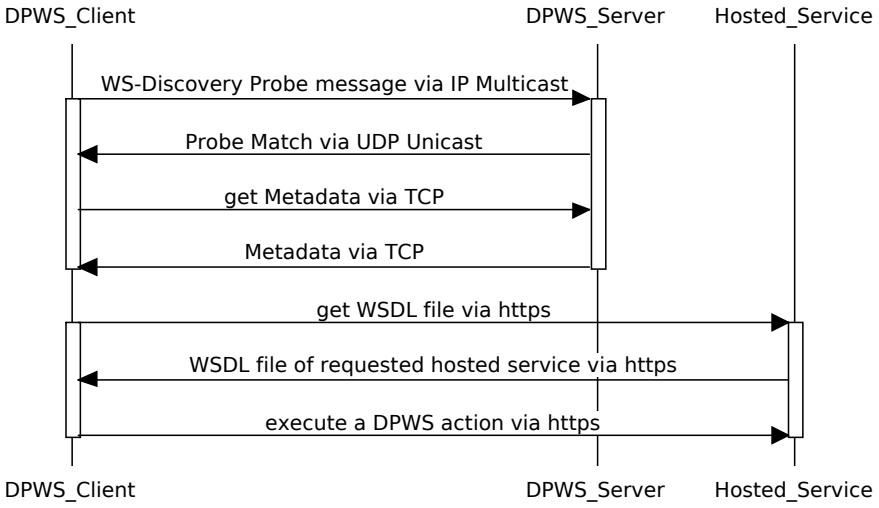

Fig. 6.11: The DPWS discovery phase

Figure 6.11 shows the standard DPWS discovery procedure. First, a client multicasts a probe message looking for a certain type of DPWS device. The server then provides minimal information about a device (DPWS probe match), which the client subsequently uses to request further metadata about hosted services on it. Each hosted service replies with a WSDL file specifying the datatypes and messages the hosted service understands, as well as the actions implemented by the service. This WSDL file can also be sent via an HTTPS connection and section 8.2 presents our approach for also securing specific DPWS actions.

The Internet Gateway Device

In UPnP an Internet Gateway Device[5] describes a standardized service that allows learning the public IP address of a middlebox, as well as adding, altering and removing port mappings. An application that supports the IGD profile is able to discover available IGDs in the network and to forward certain ports allowing the application to communicate across a NAT device. We adapted the specification of UPnP IGD to DPWS and created a secure way of establishing port forwarding entries in a DPWS enabled middlebox.

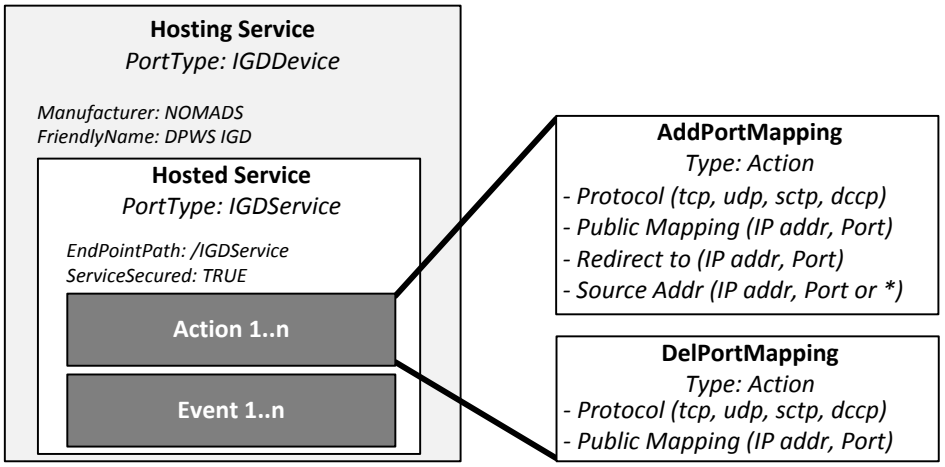

Fig. 6.12: DPWS-IGD architectural overview

Figure 6.12 shows the architecture of our DPWS-IGD. The hosting service *IGDDevice* provides the actual IGDService and its actions. The endpoint */IGDService* can be used by DPWS clients to communicate with the hosted service. *ServiceSecured* indicates that all actions belonging to this service should be invoked within a secure connection. The hosted service implements two actions: *AddPortMapping* and *DelPortMapping*. For adding a port mapping entry we extended the capability of UPnP and allow specifying a redirection endpoint and (optionally) also a filter that only forwards packets coming from a specific source transport address (*Source Addr*). This allows using the DPWS-IGD with all of our service categories as described in section 5.3.

Implementation

The DPWS-IGD was implemented using the Java Multi Edition DPWS Stack JMEDS[6] that also supports embedded devices such as home routers. JMEDS provides the necessary functionality for discovery, eventing and security and allows adding customized hosted services. An internal mapping table is used to maintain entries for dealing with duplicate requests, collisions and for exporting a list of currently active mappings. Wildcards are supported for remote hosts and the actual mappings are created using *iptables*[7]. This prototype demonstrates the functionality of our DPWS-IGD and supports to securely add, remove and list port mapping entries.

[5] http://upnp.org/specs/gw/igd2/
[6] http://sourceforge.net/projects/ws4d-javame/
[7] http://www.netfilter.org/

Evaluation

For the evaluation of DPWS-IGD we used two standard Linux PCs directly connected via a 1Gbit/s link and a round trip time of approx. 1ms. Our goal was to measure the latency that is introduced by adding and deleting a new port forwarding entry. For the first test we established 200 random mappings and measured the time for creating a new one. Figure 6.13 shows our results. The diagrams depict the minimum, maximum and average values for each experiment and the boxes represent the 80% range of all results.

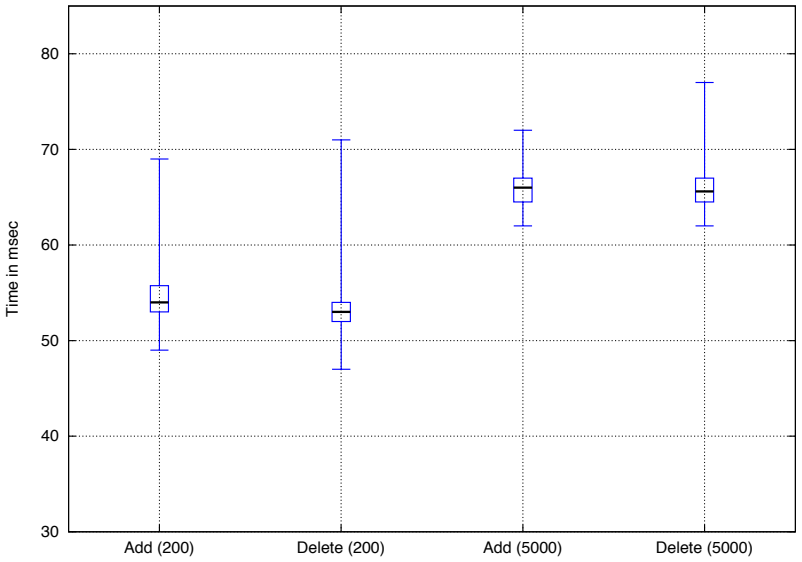

Fig. 6.13: Performance of DPWS-IGD actions dependent on existing mappings

For adding a new port forwarding entry a mean value of $54ms$ was measured. Deleting an entry took $53ms$ in average. For the second test we established 5000 mappings and again measured the time for creating and deleting a new one. The latency was only marginally higher due to the necessary check for collisions: $66ms$ and $65.5ms$ in average. To summarize, the performance tests show that DPWS-IGD scales up to a typical number of forwarding entries and only introduces minimal latency when establishing a connection. The advantage over solutions such as hole punching is that a mapping only has to be created once when using wildcards for the source addresses (stream independency).

UPnP-IGD Conclusion

UPnP offers a simple solution for automatically forwarding a port to an internal host. Once created, all hosts in the public Internet are able to send packets to the created endpoint. However, UPnP provides no security which enables hosts within the private Internet to create arbitrary port forwarding entries. Malicious software such as a trojan may create a mapping allowing other hosts to connect and control it. We target this problem by the design and implementation of a UPnP-like Internet Gateway Device that is based on the Devices Profile for Web-Services. DPWS supports digital certificates for creating and controlling port forwarding entries and by also allowing to restrict the access to a mapping for specific source addresses, DPWS-IGD is a powerful solution for all of our middlebox traversal service categories.

6.5.2 Autonomous Middlebox Traversal

As described in section 5.2.2 traditional hole punching techniques require a third party, such as a rendezvous service and a STUN server, for predicting and coordinating ports and IP addresses. This increases the complexity of the software and introduces an unwanted dependency on external infrastructure. Furthermore, it may also cause privacy and security problems, e.g. when using anonymizing peer-to-peer networks such as TOR [37]. In [100] we described a hole punching technique based on ICMP that removes this dependency and enabled establishing a direct connection between two hosts. We call this technique AutoMID: Autonomous Middlebox Traversal.

Approach

The general idea is based on the fact that once the time to live (TTL) field of an IP packet expires, the intermediate router notifies the sender with an ICMP TTL exceeded message. Parts of the headers of the original packet are used as the payload of the TTL exceeded message. Since our ICMP field test results as presented in section 4.6.4 revealed that most middleboxes allow and forward manipulated ICMP messages, this led to the approach as depicted in figure 6.14.

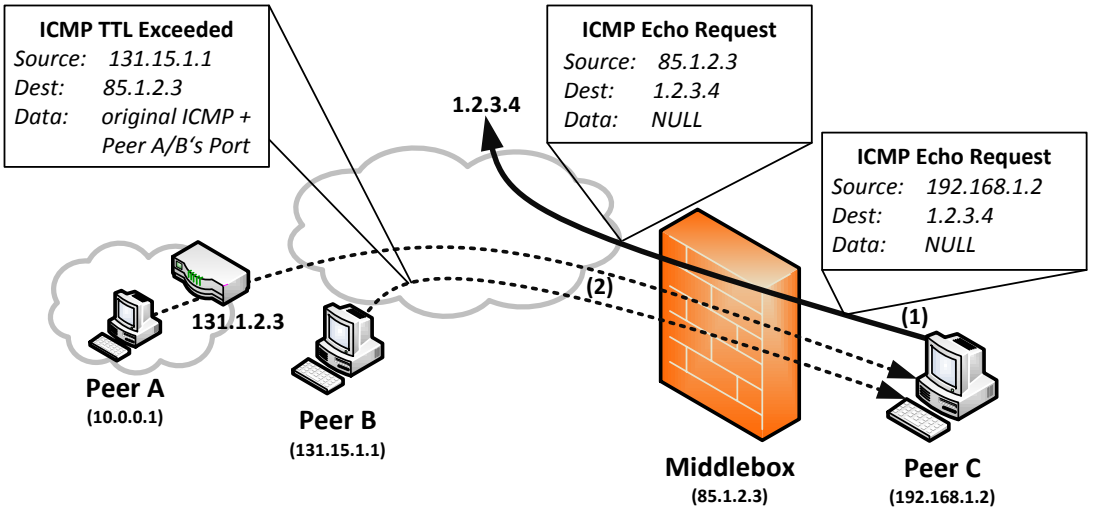

Fig. 6.14: Technical approach of AutoMID

Instead of registering with a signaling infrastructure peer C (that wants to be reachable behind a middlebox) periodically sends out ICMP Echo Request messages to a pre-defined non-existent (or controlled by the user) public IP address. These messages create a state in the middlebox and arbitrary hosts are allowed to send ICMP TTL exceeded messages to it. As depicted in step (1) of figure 6.14 the middlebox translates the source IP address of the initial ICMP packet to the public IP address of the middlebox. Thus, our approach requires that peers need to learn the public IP addresses of other peers before contacting them. This can be done be previous sessions or by a third party. However, the third party is not needed when actually establishing a connection and this is the main difference to the state of the art.

Scenarios

In the first scenario peer B, located in the public Internet, wants to establish a connection to peer C located behind a stateful middlebox. In order to reach peer C, B fakes an ICMP TTL exceeded message with it's own source IP address and peer C's public IP address as the destination. As valid ICMP TTL exceeded messages carry a copy of the original (expired) packet in their payload, peer B also has to fake the corresponding fields and include them in the payload. This requires knowledge about the destination address of the original ICMP echo request (here 1.2.3.4) as described above. The ICMP TTL exceeded messages will reach peer C, which learns about peer B's IP address (the source IP address of the packet). If an application that implements AutoMID uses pre-defined ports, peer C is now able to establish a UDP or TCP connection to peer B using the provided IP address. If the ports are not known in advanced, peer B is able to specify a port number within the payload of the TTL exceeded message as depicted in step (2) of figure 6.14 and (according to our field test results) chances are high that the ICMP packet will still be delivered.

In the second scenario both peers are behind a stateful middlebox (peer A and peer C in figure 6.14), which arises further complications. The first requirement is that peer A's middlebox allows outgoing ICMP TTL exceeded messages without seeing any related packet. Unfortunately, our field test revealed that only approx. 6% of all tested middleboxes forward such messages. However, if the ICMP message was transmitted successfully, both peers know about the others public IP addresses and are able to send messages to each other to initiate hole punching. If both middleboxes implement port preservation (probability for this constellation is approx. 50%) these messages will create the appropriate holes in the middleboxes and eventually reach the remote peers. However, if one of the middleboxes does not preserve port numbers, hole punching only works if at the same time the filtering strategy is unrestrictive enough to let the packets pass. As a possible extension a third party STUN-like server can be used to predict external ports and the client is again able to provide the predicted port within the ICMP TTL exceeded message. Again, the STUN-like server is not needed for signaling, but only on one side for querying and predicting ports.

Implementation

[100] presents three implementations of AutoMID: the algorithms for testing allowed packet sequences are integrated into our measurement suite as presented in section 4.3.1, pwnat[8] is a standalone implementation of AutoMID and GNUNet[9] also implements the approach as a transport plugin. The disadvantage when implementing the approach as described above is that all of the peers need superuser privileges to send fake ICMP messages. As a possible alternative to sending ICMP echo request messages to a fixed IP address, peer C may send UDP messages to a fixed IP address and port in order to create a mapping. Other peers then send ICMP TTL exceeded messages to this IP address and embed the original UDP packet in the payload. In this case not only the IP address of peer C has to be known, but also the public port number that is used for the UDP packet, which is possible for port preserving implementations. For middleboxes that implement an independent filtering strategy the initial UDP packet already creates a hole that can be used as an alternative way of contacting the internal peer.

[8] http://samy.pl/pwnat/
[9] https://gnunet.org/

Evaluation

AutoMID was evaluated using our field test as presented above. We integrated the necessary packet sequences into our libpcap-based test to predict the success rate of the ICMP-only and the UDP-ICMP implementation.

Component	Success Rate
ICMP-only Implementation	
Echo-Server: echo request out, TTL exc. in	48.3% for modified payload
Echo-Client: TTL exceeded out	6.02%, 5.87% for modified payload
UDP Implementation	
UDP-Server: UDP out, TTL exceeded in	78.21%
UDP-Client: TTL exceeded out (UDP)	7.43%, 6.1% modified
Port Preservation	71.45% for UDP
	51% for two arbitrary hosts

Tab. 6.3: Experimental evaluation of AutoMID

Table 6.3 shows the results. For the ICMP-only variant the Echo-Server implements the server part. Here, the middlebox has to support incoming TTL exceeded packets as a response to an outgoing echo request, which was possible in 48.3%. The Echo-Client implements the counterpart that sends ICMP TTL exceeded messages in order to signalize a new connection. If this host is located in the public Internet (scenario 1 as described above) the success rate is only dependent on the Echo-Server. For scenario 2 (both peers are behind a stateful middlebox) the middlebox needs to allow outgoing ICMP TTL exceeded messages without having seen any incoming packets. This is possible in 6.02%. If we also want to include the peer's port in the payload, the success rate drops to 5.87%.

For the UDP-based implementation the server part needs to accept incoming ICMP TTL exceeded as a response to outgoing UDP packets. This was possible in 78.21%. The client part embeds the UDP header in the ICMP TTL exceeded packet, which was successful in 7.43%. As stated above, the UDP implementation requires the middleboxes to implement a port preserving binding strategy, which is true for 71.45% of a single hosts and for approx. 51% for two arbitrary hosts in the Internet.

AutoMID Conclusion

Autonomous Middlebox Traversal (AutoMID) utilizes ICMP messages to enable hole punching without the need for a third party. We covered two application scenarios and presented two technical approaches. AutoMID works reasonably well if only one host is located behind a stateful middlebox. Since many middleboxes don't allow outgoing ICMP TTL exceeded messages, the success rate for scenarios where both hosts are behind stateful middleboxes is rather low. Nevertheless, AutoMID can be seen as an additional technique that extends the set of available middlebox traversal techniques and can be seen as a valid choice if a policy forbids the use of third parties.

6.6 Evaluation and Discussion

When presenting state of the art middlebox traversal techniques in section 5.2.4 we identified eight criteria that were used to evaluate existing solutions. We stated that all of the existing techniques have their strengths, but also significant weaknesses. When evaluating the NOMADS framework using the same criteria we get the following results:

The **performance** criteria was split into two branches: First, we looked at the performance of the framework when establishing a new connection. When using signaling for coordinating hosts, a latency will be introduced. In NOMADS the latency through signaling is minimized using the knowledge-based approach that only requires two signaling messages when using the standard service category SPS. A time-consuming pairing process like the one in ICE is not needed. The second branch dealt with the performance of the actual data connection, which depends on the middlebox traversal technique that is applied to the situation. NOMADS always tries to establish direct connections by parameterizing hole punching and it also considers the role of the application, e.g. how many consecutive connections will be made from different source addresses (stream independency). A detailed performance evaluation is done in the following section.

Security describes the capability of the middlebox traversal solution to restrict a traversal solution to authorized users, thus taking decisions of external entities, such as administrators, into account. In NOMADS there are two possibilities: First, the service category Secure Service Provisioning defines that only authorized users are allowed to create and access a mapping. The knowledge-based approach takes care of selecting an appropriate technique and the signaling protocol includes the possibility to authenticate hosts before establishing mappings. Second, NOMADS supports the definition of policies for restricting the use of certain techniques or to prevent an application from establishing a mapping. For example, a malicious application will not be able to utilize our framework (like in UPnP) unless it is registered to the Session Manager. This is an important step towards our goal that middlebox traversal should enable a communication in a way that is "compliant with authorization and accessibility requirements, as well as additional policies as defined by the user or the administrator of the network" (chapter 5). A framework for managing secure identities is introduced in the following chapter.

The number of **Supported Techniques** in NOMADS is not limited. Once new techniques become available they can be integrated into the Middlebox Traversal Module. This requires to also set a policy that defines the applicability condition for the corresponding middlebox traversal service category. So far, NOMADS integrates the most popular techniques and parameterizes them based on the current behavior.

We defined **Flexibility** in the context of middlebox traversal as the ability to support many applications and to adapt to the current situation. Since NOMADS supports many existing solutions and is able to control and adapt them further, it provides a great flexibility to existing applications.

Many state of the art solutions have no **Support for legacy applications**. If the technique is not built into the application it only helps if e.g. the operating system allows setting a global rule (SOCKS). For NOMADS we designed two application interfaces both located at the Application Interface Module: First, a socket API for newly developed applications and second, a TUN-based approach allowing legacy applications to utilize the framework without any changes to the code.

The dependency of **External Entities** may limit the success rate of a traversal solution if the entity is not available or if it only offers poor performance (e.g. data

relay). Many existing solutions rely on external third parties for coordinating traversal techniques. NOMADS is also dependent on an external signaling infrastructure (if signaling is used) and the individual techniques show the same dependency as if they were used standalone. However, due to the large set of integrated techniques, the dependency to a specific external entity is rather low. Furthermore, NOMADS can also be used in unilateral deployments where no signaling is required. Using a decentralized signaling infrastructure such as P2PSIP also helps to minimize this dependency. Finally, in section 6.5 we presented our new middlebox traversal technique AutoMID that allows hole punching without the need for a third party, removing the dependency for external entities.

Control-based techniques require to install **Software on the Middlebox**. If a middlebox on the path to the public Internet does not run it, the technique cannot work. NOMADS integrates many solutions and chooses the most applicable one from the set of available techniques. Thus, if a particular control-based solution is not available due to missing software, it is able to choose a different (behavior-based) technique. Our Session Manager that was designed to run on the middlebox, can easily be shifted towards the end-host and does not introduce such dependency.

Techniques that use signaling for coordination require to install **Software on end-hosts**. When using the signaling protocol in NOMADS this is also true. In contrast, NOMADS also supports unilateral deployments where only one end-hosts runs the framework. This however limits the number of applicable middlebox traversal techniques since many of them require knowledge about the potential requester.

6.6.1 Adaption to Experimental Results

The results of the field test as presented in section 4.6 serve as a motivation for our knowledge-based approach. In section 4.6.8 we presented our lessons learned and gave recommendations on how to improve future middlebox traversal techniques to increase their success rate. This section shows how we reacted to our findings and how NOMADS targets each of them.

The first finding was that the STUN algorithm is actually wrong for 25% of all Full Cone NATs and 5% of all Symmetric NATs when determining the independency of the NAT binding. This leads to low success rates for hole punching and other techniques where an external endpoint needs to be predicted. In NOMADS we follow the recommendation not to differentiate between endpoint independent bindings and connection dependent bindings, but to differentiate between the possibility to predict an endpoint or not. For predicting external ports we integrated different independent variations for endpoint prediction into the endpoint prediction part of the Middlebox Analyzer and are thus able to make a clear statement about the applicability of a traversal technique.

The second finding revealed that middlebox traversal often fails for two hosts both behind NAT because only 61.4% (56.6% for TCP) of all constellations provide the necessary prerequisites for establishing direct connections. We proposed to also consider other constellations by understanding their behavior and by introducing a technique called “swap role”. NOMADS considers the behavior of both endpoints and allows coordinating them through signaling. After analyzing their behavior and pairing their model instances NOMADS is able to swap the roles of the requester and service if necessary and establish the connection in the other direction.

The third finding showed that although many middleboxes implement a connection dependent binding strategy, endpoint prediction is still possible by analyzing binding

and mapping patterns. NOMADS implements this combined endpoint prediction algorithm as part of the Middlebox Traversal Module, which is considered by the requester (when using signaling) and the service (when making a decision). According to our field test this endpoint prediction method is successful in 91% for UDP and 86% for TCP, resulting in a much higher success rate compared to state of the art techniques. When also considering error-prone mappings and by sending multiple simultaneous packets these numbers can even be increased further.

Our forth finding showed that the success rate of a hole punching algorithm depends on the behavior of the involved middleboxes and when adapting certain parameters (e.g. the TTL) to the current situation the success rate can be increased. NOMADS parameterizes hole punching algorithms dependent on the behavior of the involved middleboxes by pairing their model instances. Endpoint prediction as well as an unwanted deallocation of a created mapping are essential parts of the hole punching algorithms controlled by the Decision Module.

Finally, the last finding stated that Large Scale NATs (and cascaded NATs) can already be found in today's networks and are another reason for the poor success rates of existing behavior-based traversal techniques. The success rate of hole punching can be increased in many cases by setting the TTL of the hole punching packet to a value smaller than the number of hops to the first stateful middlebox of the requester. Without considering multiple layers of middleboxes this value is usually set to 2. In NOMADS we integrated our topology detection algorithm that reveals parts of the network topology and detects stateful middleboxes. This allows setting the TTL of the hole punching packet accordingly and also makes sure that control-based techniques such as UPnP are only applied if all middleboxes on the path support the technique.

6.6.2 Applicability Evaluation

In section 6.4.1 we presented the scenarios that are addressed by our framework. Voice over IP, peer-to-peer applications and traditional services are just a few examples for applications in unmanaged networks that suffer from the existence of middleboxes. Future scenarios that require a coordination of home networks for providing resilience, privacy or just a decentralized way of sharing data in a secure and independent manner will even aggravate the problems. Additionally, the deployment of middleboxes at the provider side (e.g. LSN) and the presence of multiple layers introduces new complications and breaks existing state of the art solutions for middlebox traversal. NOMADS targets these problems by supporting different middlebox service categories for applications and by thoroughly analyzing the actual behavior and applicability of a traversal technique. Our TUN-based application interface supports legacy applications and can already be used today. Finally, new middlebox traversal techniques, DPWS-IGD and AutoMID, address new requirements such as privacy and security and help to extend the set of available solutions to be integrated into a frameworks such as NOMADS.

For mobile networks we distinguish between LTE-based home networks in remote areas and mobile clients that usually use resource restricted devices to access the network. For LTE-based home networks the requirements and solutions as stated above are still the same. However, the chance for cascaded middleboxes in these networks is very high, thus making state of the art algorithms even worse. Since providing services on smartphones is rather unusual, the main application area for mobile users are applications such as Voice over IP and other applications that enable new services by establishing direct connections between devices. Since provider-based middleboxes are very common in mobile networks, again the topology analysis, as well as the parameter-

ization of existing techniques are important contributions of this thesis. Additionally, many users of unmanaged networks may wish to use their mobile devices as a frontend for connecting to services in their network. In this case the mobile device is the requester of a session and should therefore be able to also take part in the coordination process. A first lightweight implementation for Android allowing the analysis of the current network topology was described in section 4.5.3.

For corporate networks NOMADS allows defining policies to prevent the allocation of public endpoints for unauthorized users. Furthermore, the service category Secure Service Provisioning makes sure that only traversal techniques are used that are compliant with the policy as defined by the administrator. Here, not only traditional and future Network Address Translation devices are considered, but also the session-based control of firewalls and other middleboxes. Instead of manually asking an administrator to configure a firewall, NOMADS is fed with a policy that allows incoming connections for a configurable amount of time. Once the time expires the session is closed and a new one cannot be established without further authorization. NOMADS mechanisms for the policy-based configuration of dynamic sessions also help with existing control-based solutions in corporate networks. One last missing component is an easy to maintain, but yet secure infrastructure for managing digital certificates and trust relationships. A security infrastructure for unmanaged networks is therefore presented in chapter 7.

6.6.3 Performance Evaluation

Frameworks that utilize signaling for coordination before actually establishing a connection introduce an additional delay to the network. State of the art middlebox frameworks, such as ICE (see section 5.2.3), collect information about the network, exchange it via a third party and determine a working endpoint by testing possible combinations. Our knowledge-based approach does not require to actively (meaning testing by actively sending messages) pair possible endpoints. Instead, the Decision Module instantly decides which technique should be used, invokes it and provides a working endpoint to the requester. This section compares NOMADS to the state of the art framework ICE and analyzes if the decoupling of the knowledge-gathering process from the connection establishment actually poses a performance advantage.

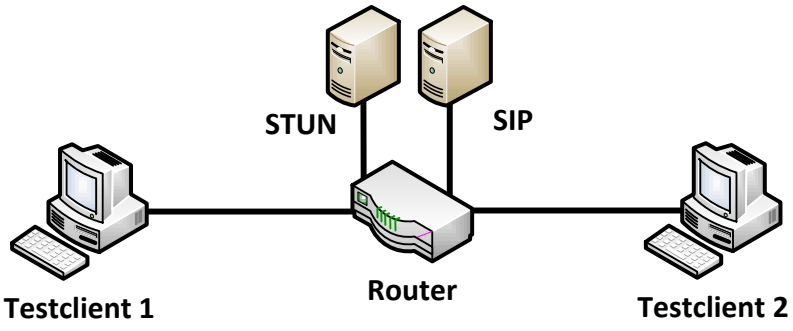

Fig. 6.15: Setup for our performance evaluation

In order to experimentally measure the introduced delay for NOMADS and ICE, we configured a testbed as depicted in figure 6.15. Two clients with an Intel Core i5-2520M CPU and 8Gb of RAM and two Asus EeeBox PCs with an Intel Atom CPU and 2Gb of RAM were connected to a 100MBit/s router running the OpenWRT operating system. All hosts were running a standard Debian GNU/Linux 6.0 operating system.

The clients represented the NOMADS and ICE instances (requester R and service S) and the Atom-based PCs hosted a SIP signaling infrastructure, as well as a STUN server (opensips[10] and stund[11]). The router allows introducing configurable delays to the network by utilizing the *ip_queue* element of *iptables*[12]. There was no cross-traffic and the average roundtrip time of ICMP echo requests/responses between two arbitrary hosts was $1.3ms$. We measured the time on the requester side from the beginning of the connection establishment/signaling (received packet on the TUN device for NOMADS and beginning of the ICE session for ICE) until the first byte (e.g. a TCP-SYN) was sent out. Each test run was repeated 50 times and the largest standard deviation for NOMADS was 4.76 and for ICE 1.2 with normal deviation.

For NOMADS the measurements include the following packets: the requester queries the STUN server (1 RTT, 2 packets: binding request and response) and sends the service request via SIP (1 RTT, 2 packets: SIP message and SIP OK). The service allocates the mapping using a parameterized hole punching packet (1 packet), queries the STUN server (1 RTT, 2 packets: binding request and response) and returns the allocated public endpoint via SIP (1 RTT, 2 packets: SIP message and SIP OK) (note that many of the tasks can be done in parallel, e.g. the hole punching packet and querying the STUN server on the service side). For ICE we used the PJNATH implementation[13] and extended the *icedemo* example with SIP signaling capabilities using the same code as we were using for NOMADS (eXosIP library[14]). For the first experiments we only used one STUN-derived candidate and enabled the ICE aggressive mode that terminates the pairing process as soon as the first working endpoints are found. Table 6.4 show the results of our measurements.

Conn.	Frmw.	1.3ms	20ms	50ms	100ms	150ms	200ms
R - SIP	NOM.	37.2ms	55.1ms	87.8ms	135.9ms	182.1ms	236.3ms
	ICE	500.7ms	520.7ms	550.6ms	600.6ms	651.3ms	700.6ms
R - STUN	NOM.	37.2ms	47.9ms	78ms	128.2ms	177.1ms	228.4ms
	ICE	500.7ms	519.9ms	550.3ms	599.8ms	650.9ms	700.5ms
R - S	NOM.	37.2ms	37ms	36.5ms	36.9ms	36.8ms	36.2ms
	ICE	500.7ms	520.9ms	550.9ms	600.4ms	649.3ms	700.9ms

Tab. 6.4: The connection setup time in milliseconds dependent on the network delay

With our standard setup (RTT of $1.3ms$) NOMADS only introduces an additional delay of $37.2ms$ to the network when setting up a connection. Surprisingly, the ICE pairing process took $500.7ms$, which may be acceptable for setting up a Voice over IP connection, but may not be tolerated by many users for other applications. Since NOMADS signaling process does not include packets traveling from the requester to the service directly (only STUN and SIP are involved) the connection setup time (CST) is only dependent on the RTT to the STUN or SIP server (here we assume that both hosts use the same SIP server). Table 6.4 shows the impact of increasing the RTT on different paths.

[10] http://www.opensips.org/
[11] http://sourceforge.net/projects/stun/
[12] http://people.redhat.com/berrange/notes/network-delay.html
[13] http://www.pjsip.org
[14] http://savannah.nongnu.org/projects/exosip/

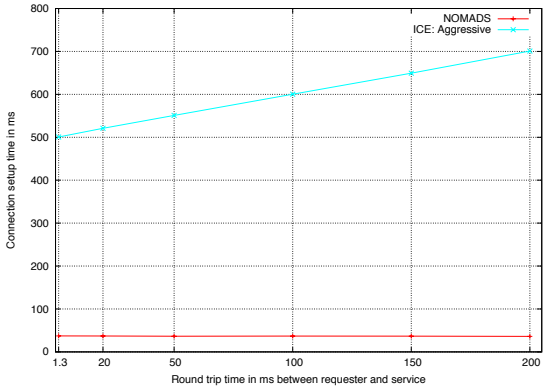

Fig. 6.16: Comparision for ICE's agressive mode

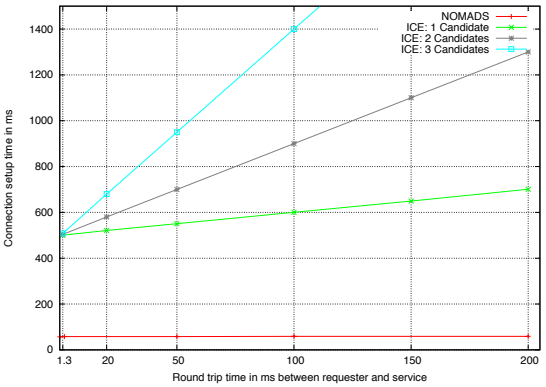

Fig. 6.17: Comparision for ICE's regular mode

Based on these results the connection setup time can be described as follows:

$$\begin{aligned} CST_{NOMADS} =& NOMADS_{core} + RTT_{R-STUN} + RTT_{S-STUN} \\ &+ RTT_{R-SIP} + RTT_{S-SIP} + RTT_{SIP_R-SIP_S} \end{aligned}$$

$NOMADS_{core}$ represents the time for processing the XML description coming from the remote host, the pairing process, as well as the packet translation between the TUN-device and the actual network interface. It is important to note that the time needed for the actual pairing process (see section 6.4.6) can almost be neglected: When comparing and emulating different middlebox constellations according to our decision tree as depicted in Appendix C.1, we measured an average pairing time of less than $1ms$, no matter how complicated the situation was. Since both the requester and the service have to look up a STUN server we have to add one RTT to STUN for each endpoint (RTT_{R-STUN} and RTT_{S-STUN}). For the signaling part, R and S both send one message to the SIP server (RTT_{R-SIP} and RTT_{S-SIP}). Additionally, the latency between the SIP proxies also has to be considered ($RTT_{SIP_R-SIP_S}$).

ICE performs STUN connectivity checks between the requester R and the service S for each candidate pair. Thus, the connection setup time is not only dependent on the RTT between R and S, but also on the number of candidates that have to be checked. Each candidate on the requester side is paired with each candidate on the service side which results in an additional delay of $cand_R * cand_S * RTT_{R-S}$. For our tests we enabled the (optional) aggressive mode that terminates the candidate pairing on the first successful pair, thus delivering the fastest result possible[15] and setting $cand_R$ and $cand_S$ to 1. We can give the following formula where ICE_{core} includes the message processing and the delay to the STUN server used during the gathering process:

$$\begin{aligned} CST_{ICE} =& ICE_{core+STUN} + RTT_{R-SIP} + RTT_{S-SIP} \\ &+ RTT_{SIP_R-SIP_S} + cand_R * cand_S * RTT_{R-S} \end{aligned}$$

Figures 6.16 and 6.17 depict the advantage of the NOMADS framework. While the connection setup time of ICE depends on the latency between the requester and the receiver and (in regular mode) also on the number of candidates, NOMADS is able to instantly invoke a middlebox traversal technique. This dramatically decreases the number of messages and therefore the overall connection setup time compared to the state of the art.

[15] it is important to note that ICE may not determine the optimal result in aggressive mode since not all candidates are checked.

6.7 Summary and Key Findings

This chapter shows that the knowledge about the behavior of involved middleboxes as gathered in chapter 4 can be used to apply and parameterize middlebox techniques suitable for applications and the involved infrastructure and therefore answers question Q2 according to section 1.1. We presented our knowledge-based approach to middlebox traversal NOMADS that decouples the actual connectivity establishment and periodically scans the network to gather knowledge about the behavior and presence of middleboxes. Whenever a decision about middlebox traversal is needed the knowledge is combined with local policies and requirements of the applications to parameterize an appropriate middlebox traversal technique. A signaling protocol helps to coordinate NOMADS instances and application interfaces support legacy as well as new applications. Obtained results confirm the applicability to existing problems and an increased performance compared to state of the art frameworks. Finally, we presented two new middlebox traversal techniques, DPWS-IGD and AutoMID, that extend the set of available middlebox traversal techniques and provide solutions for restricted environments.

Key Findings of this chapter
(and contributions according to section 1.2):

C6.1a **Knowledge-based middlebox traversal was proposed to react to the findings of our field test.**

C6.1b **The NOMADS framework supports our middlebox traversal service categories and considers external user-defined policies.**

C6.1c **NOMADS integrates existing traversal techniques and parameterizes them based on the actual behavior to increase their success rate.**

C6.1d **NOMADS is significantly faster and more applicable to open problems than state of the art frameworks.**

C6.2 **Two new middlebox traversal techniques for restricted environments were developed: DPWS-IGD can be seen as a secure alternative to UPnP and AutoMID allows hole punching without the need of a third party.**

Part IV

SECURITY AND APPLICATION OF MIDDLEBOXES

7. A SECURE SERVICE INFRASTRUCTURE FOR UNMANAGED NETWORKS

7.1 Introduction

Today, the authentication of services in the Internet is provided via digital public key certificates following the IETF X.509 standard [28]. The Public Key Infrastructure (PKI) model consists of a hierarchy with global Certificate Authorities (CAs) issuing certificates to services and a chain of trust for verifying them. Due to the complexity of today's PKIs, client authentication is rather done via passwords instead of also using digital certificates and mutual authentication as defined in protocols such as SSL/TLS [34]. Maintaining the Public Key Infrastructure for servers is already a huge challenge and inexperienced users are not able and willing to participate in this process [72].

However, home and small company networks would benefit from having enterprise grade security mechanisms based on X.509 certificates as their mechanisms allow to establish closed groups. In section 5.3 we introduced service categories for applications and argued that a middlebox framework should take these categories into account when making decisions about middlebox traversal. The category Secure Service Provisioning relies on secure identities to authenticate the requester and to prevent unauthorized access to a service. Other examples for the direct communication between two users are VoIP and applications such as Android Beam that utilize Near Field Communication (NFC) for sharing data between two devices. Additionally, future home networking scenarios make the secure (remote) access to devices and services in the home, as well as home-to-home communication highly desirable.

For inexperienced users that are not able and willing to maintain a PKI, the deployment and management of digital certificates must be dramatically simplified. Thus, this chapter proposes a secure service infrastructure for unmanaged networks that assists inexperienced users to maintain their own PKI. The infrastructure hides technical complexities from its users and provides assistance functionalities and semi-automatic processes reaching from the discovery of an appropriate certification server to the trust establishment across multiple networks. The centerpiece of our solution is called MicroCA, a small box that integrates discovery, authorization and authentication services.

As described in section 6.4.1, unmanaged networks are not professionally administrated, e.g. home and small company networks. In home networks the access to the wireless network is usually the only security mechanism that is available. Other typical home services such as UPnP [50] implement no security at all, mostly because of the lack of secure identities in home networks. Small company networks would also benefit greatly from a user-friendly security architecture, as they often lack a professional administrator. For example, start-up companies may not have the capacities to take care of securing their devices and to set up a secure network in a professional way. Our MicroCA provides an out-of-the-box solution with assistance functionalities for setting up a secure network. Finally, our solution also targets administrators of professionally managed networks by relieving the administrators and giving them more time to work on other problems.

7.2 Survey of the State of the Art

This section shortly presents the state of the art regarding identity management in unmanaged networks and two different trust models for the creation of a binding between a public key and its owner.

7.2.1 Identity Management for Unmanaged Networks

Identity management describes the authentication and authorization of identities to services in a network. Authentication aims at verifying the identity of an entity (e.g. a host), whereas authorization deals with the assigned access rights. Besides using passwords, a common approach for authentication is to issue a digital public key certificate to a user for mapping an authentication credential (here the public key) to the actual identity.

There have already been a few approaches for introducing a PKI to unmanaged networks. [89] propose a PKI-based home device authentication mechanism using a hierarchical PKI structure. A Home Registration Authority (HRA) asks an external CA for digital certificates on behalf of its devices. We see two problems with this approach: First, keying material and certificate signing requests are computed by an external CA that is not part of the home network itself, which introduces privacy issues, because the disclosure of devices to the external CA might not be desired. Second, the hierarchical CA infrastructure has to be maintained by a third party, thus introducing costs and management overhead.

A concept for a personal CA for certification of devices inside a PAN (Personal Area Network) is presented in [95]. This approach is limited to devices belonging to one network only. The interconnection between PAN domains is not possible. The authors assume to already have connectivity to the registration service and do not target the registration and bootstrapping process.

As a decentralized identity management approach, SecBook [36] publishes keying material via facebook and uses social structures to establish trust in a public key. Facebook users sign their friend's keys and use them for securing their communication. Another community-based approach is Monkeysphere [148] that allows validating a certificate of a website based on user-signed ratings. A browser plugin passes the request to a validation agent, which then replies back with the rating for this website.

7.2.2 Trusted Third Party

The trust anchors of a centralized PKI are Certification Authorities (CA). CAs map a public key to an identity by issuing digital certificates. In order to verify an identity, a client needs to trust the CA, obtain the public key from it and check the signature. Since most users are not able to differentiate between good and bad CAs, the public keys of many CAs are included in operating systems or software products such as web browsers. Thus, trusting a CA often means trusting the browser or operating system. If a CA is not known to the operating system or browser it generates a warning, which is often ignored and clicked away by the user.

This indeed is the weakness of many PKIs as we have them today. A recent analysis of the SSL landscape shows that "the X.509 certification infrastructure is, in great part, in a sorry state" [72]. The authors revealed that only 18% of all certificates of the Alexa top 1 million list[1] are accepted by today's web-browsers without any warning. Reasons

[1] http://www.alexa.com/

for invalid certificates are incorrect or missing hostnames, incorrect certification chains or expired certificates.

However, a CA provides a mapping of identities to an issuer and if the issuer is known and trusted, identities can be easily verified and used for authentication. In fact, many companies implement their own PKI and maintain it as a closed system, thus circumventing the disadvantages of the distributed PKI as used in the web. We therefore propose to use the concept of a CA for managing identities within our unmanaged networks and implement an assistance system to overcome the difficulties and technical complexities that exist in managing certification.

7.2.3 Web of Trust

Another approach for a trust model is the Web of Trust (WoT). Here the identification of a user and the mapping of the user to a specific public key is based on one or more signed assertions created by other users. The public key of the user and the signed assertions are published on a number of well-known key servers. An entity that wants to authenticate a user based on a public key obtains the user's public key and signatures from the key server and assesses the authenticity of this public key by verifying the signatures. Authentication based on public keys taken from a WoT-like infrastructure are mostly used for authenticating email communication. The drawback of these systems is that the mapping between an identity and a key is only done via names and email addresses, which can be easily mixed up ("Which John Smith?"). Additionally, users who have multiple identities or devices need to maintain each single key separately, which on the one hand adds security but on the other hand imposes important usability limitation.

A recent analysis of the WoT shows that "the WoT is likely to be quite an effective PKI structure within smaller node neighborhoods, and particularly for those users that frequently sign other keys and are active in the WoT" [155]. Since we assume that the interconnection of unmanaged networks, such as home networks, is done within smaller neighborhoods with much cross-signing, we propose to use the Web of Trust for establishing trust relationships between different MicroCAs by letting them cross-sign each other.

7.3 Architectural Overview and Components

Our approach for a security infrastructure for unmanaged networks is based on a hybrid solution combining the advantages of centralized and decentralized approaches as presented in the last section and shown in table 7.1. Within one network under one administrative control (we assume each network belongs to a user that is responsible and capable of deciding which users and devices also belong to the same network) a local Certification Authority, which we call MicroCA, issues certificates to entities to express their binding to that network. Based on these certificates, users within one network are able to authenticate each other. For authenticating users belonging to different networks, thus holding certificates issued by another MicroCA, the public key of the remote MicroCA has to be known and considered trustworthy, which is done using the WoT approach.

The following sections describe the building blocks of our security architecture. We first present requirements and contributions (section 7.3.1) and give an overview of the MicroCA (section 7.3.2). Afterwards we present the user-friendly identity management (section 7.4), as well as the interconnection of different MicroCAs (section 7.5).

	Centralized CA	Web of Trust	MicroCA
Security	(+) depends on CA dep. on browser/OS	- depends on user difficult to verify fake IDs difficult to detect	+ hybrid trust into key fam. easy to verify
Usability	(+) ok for clients setup complicated	– cross-signing hard to understand	+ assistance mechanisms
Costs	– high	+ low	+ low

Tab. 7.1: Comparison of possible trust models

7.3.1 Requirements and Contributions

The following table 7.2 depicts functional and non-functional requirements, as well as the contributions of our security infrastructure. The table also lists the sections that describe the individual contributions in more detail.

	Requirement	Contribution	Section
Functional Requirements			
R1	Secure and unique IDs	Cryptographic Identities	7.4.1
	Every entity must possess a unique cryptographic identifier. Our IDs are derived from public keys and concatenated with the MicroCA's ID.		
R2	Authentication	MicroCA as trust anchor	7.3.2
	Authentication of entities within the same network is required. Our MicroCA acts as a trust anchor for its network.		
R3	Interoperability	Pairing Mechanism	7.5
	Establishing trust into remote networks is required. Our pairing mechanisms allow exchanging certificates to authenticate remote devices.		
R4	Security	MitM protect. and HW extension	7.4.2, 7.7
	Common attacks have to be prevented and keying material has to be protected. Our PIN-based approach for registering devices and hardware security mechanisms for protecting keys solve these issues.		
R5	Authorization	Policy Manager and PEP/PDP	7.6, 8.2
	Mechanisms for defining and enforcing access rights are needed. We allow connecting SSL/TLS enabled services to our framework by intercepting the SSL/TLS handshake. Section 8.2 shows a reference example based on XACML.		

Non-Functional Requirements			
R6	Usability	Assistance Systems	7.4.2
Security mechanisms must be easy and intuitive to use. Our mechanisms hide the technical complexity from inexperienced users.			
R7	Security per Default	No Opt Out	7.4.2
Security mechanisms must be mandatory for users. In our infrastructure a connection is only possible for registered entities.			
R8	Support for legacy Appl.	SSL interception	7.6.1
Access to legacy services must be possible. In section 8.2 we show how our infrastructure can be used by legacy applications supporting the SSL/TLS protocol.			

Tab. 7.2: Requirements and contributions of our security infrastructure

7.3.2 Components

Figure 7.1 shows the components of the MicroCA including interfaces for the users and the "administrator". The central component of the MicroCA is the Entity Directory, which holds the identity and security policy database. It stores information about registered devices, issued certificates, their access rights, as well as trust relationships to remote MicroCAs. All other entities provide interfaces to the Entity Directory in order to access and modify it. Therefore, it is possible to define additional arbitrary interfaces, such as the XACML policy interface, as depicted in figure 7.1. A reference example on how to connect XACML to our MicroCA is given in section 8.2.

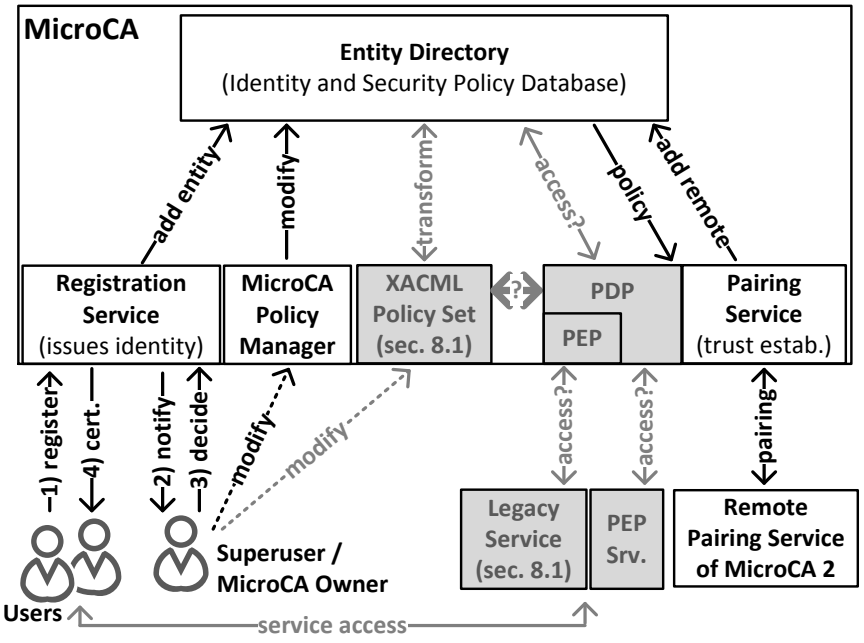

Fig. 7.1: Architecture of the MicroCA

The owner of the network, or administrator, has a special role. He monitors and supervises activities and decides about device registrations, access policies and trust relationships to other MicroCAs. However, easy to use interfaces hide the complex technical background in order to also allow inexperienced users to hold the administrator role. In the following sections we explain the structure of our identities, the device registration process that issues certificates, the network pairing service, as well as our authorization approach in more detail.

7.4 Identity Management

7.4.1 Identities

Our approach for cryptographic identities is similar to the Host Identity Protocol (HIP) [98]. Each entity (user, device, service) owns a public/private keypair and derives its globally unique identifier from the public key. The identifier can then be used for expressing access rights, for sending messages or for routing to this device. Certificates for identities are issued by MicroCAs to express their membership to a network. For example, a home network runs its own MicroCA on the home router, which then signs certificates for all devices, users and services belonging to this home.

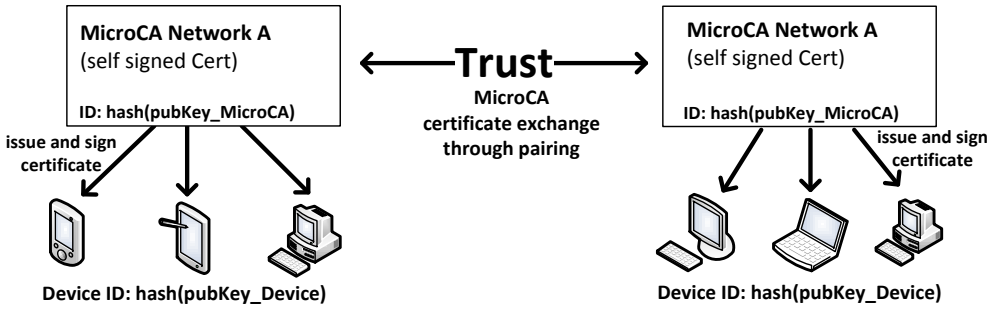

Fig. 7.2: Identities issued by the MicroCA

Figure 7.2 shows how certificates are issued to devices and how identities are derived from them. The MicroCA holds its own self-signed certificate, which may be exchanged with remote MicroCAs in a WoT-like manner in order to establish trust to other networks. The public key of the MicroCA is processed by a cryptographic hashing function and serves as the networkID. Devices use the following concatenated identifier for expressing the membership to a MicroCA:

$$\begin{aligned} ID\_MicroCA &= hash(MicroCA\_PubKey) \\ ID\_Device &= hash(Device\_PubKey).hash(MicroCA\_PubKey) \\ ID\_Service &= hash(Service\_PubKey).hash(MicroCA\_PubKey) \end{aligned}$$

7.4.2 Device Registration

One of the main and most urgent questions is how to issue, maintain and distribute valid certificates if we assume that the average user of an unmanaged network is not an expert. Instead of using rather technical terms such as *certificate* or *issue and sign*, we propose to use an understandable metaphor: **Device Registration**. In order to join a network, a user needs to register (or pair) his device with the corresponding

MicroCA. During this guided and user-friendly process the device obtains a public key certificate from the local MicroCA and the user is able to join the network. The actual certification process is completely hidden from the user.

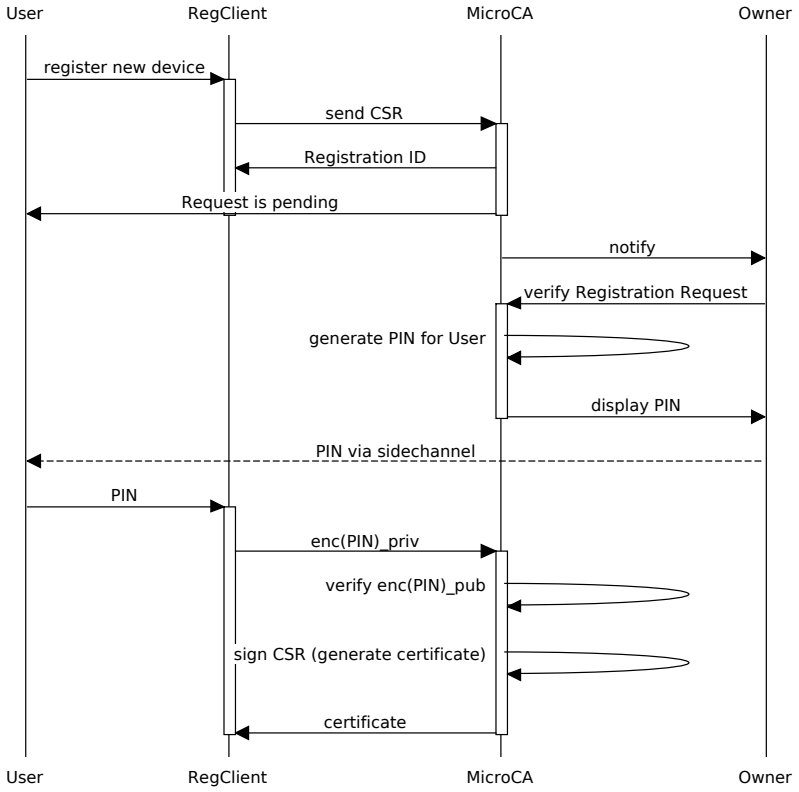

Fig. 7.3: Sequence diagram of the Device Registration Process

Before running the actual protocol, the client needs to discover and connect to the MicroCA. Here, we have to make sure that no man-in-the-middle attack (meaning an attacker introduces a fake MicroCA to the network) is possible. This can be done by performing an authenticated Diffie-Hellman (DH) key exchange with the MicroCA or by relying on near field communication techniques and benefit from the short radio distance. A WLAN approach that includes the automatic network configuration with the help of Zeroconf technology [19] will be further described in section 7.7.

The steps of the actual device registration protocol after discovering and connecting to the registration service (and thus being sure about the authenticity of the MicroCA) are shown in figure 7.3. Once the user's registration client has located the device registration service, it creates a Certificate Signing Request (CSR) and sends it to the registration service of the MicroCA. This registration request also contains meta information about the user and the device. Based on this information the owner of the MicroCA is notified and has to decide about the device registration request. In the case of a positive decision the MicroCA generates a random validation PIN code that is needed by the user to complete the registration. The PIN code is transferred to the user within a side channel, such as voice or email and acts as an implicit identity verification. This step is essential for the security of our system, because the information given in the first step of the registration process is not trustworthy and needs validation. Only if the user is able to provide the PIN code (signed with the private key: $enc(PIN)\_priv$ and verified by the MicroCA with the public key: $enc(PIN)\_pub$), the registration service

requests the MicroCA to sign the user's certificate and send it to the device. Finally, the registration service adds the identity of the newly paired device to the Entity Directory and grants default access rights. The user is now able to join the actual network, e.g. by connecting to a wireless network as described in section 7.7.

7.5 Trust Establishment

As each network operates its own MicroCA, mechanisms need to be created to also allow the verification of certificates issued by remote MicroCAs. To authenticate entities belonging to a remote network, the public key of the MicroCA that signed the certificate for the entity is needed. In the state of the art trust into a third party is established by introducing a chain of trust via multiple levels of CAs. This approach could also be adapted to our MicroCAs. However, in section 7.2 we identified the trust into an unknown CA as the weakness of today's PKIs and we further argued that centralized CAs introduce privacy and cost issues.

Therefore, we propose two additional ways of establishing trust into the public key of a remote MicroCA. In our architecture, this process is called pairing and the corresponding module is depicted on the very right bottom of figure 7.1. After a successful pairing, the foreign MicroCA is added to the Entity Directory and, if authorized, devices belonging to this network are able to access services from remote. The following sections describe our two approaches for pairing.

7.5.1 Direct Pairing

The first mechanism requires that two users, who are authorized to perform the pairing process, meet each other personally. Our identity exchange protocol then runs between their devices (e.g. via Bluetooth or Near Field Communication) and exchanges the public keys of their MicroCAs. This direct exchange is the most secure way for exchanging keys, because the mapping (public key to identity) is implicitly done by the users.

The trust exchange is depicted in figure 7.4 and is also described in [102]. First, an authenticated Diffie-Hellman key exchange prevents man-in-the-middle attacks. Both users have to verify and confirm a fingerprint to make sure the DH-parameters are correct. After the actual DH key exchange the certificates can be exchanged and verified in a secure way. The pairing of two MicroCAs satisfies three needs:

1. The MicroCA certificates are exchanged securely.
2. As the users performing the pairing know each other, they are also able to identify each other. The trust into the other person and confidence about the identity is transferred into the exchanged certificate.
3. After the pairing, both networks are able to verify certificates issued by the remote MicroCA.

7.5.2 Remote Pairing using Social Networks

In many cases it is not possible for users to meet in person. Thus, our second mechanism is based on a Web of Trust-like approach and allows exchanging the public keys of the MicroCAs via an insecure channel. One of the problems of traditional Web of Trust approaches is usability. The mapping between keys and identities is often done using an email address as an identifier, which makes it hard for an inexperienced user to decide if the mapping is trustworthy or not. For example, an attacker could publish a fake key by using a very similar email address.

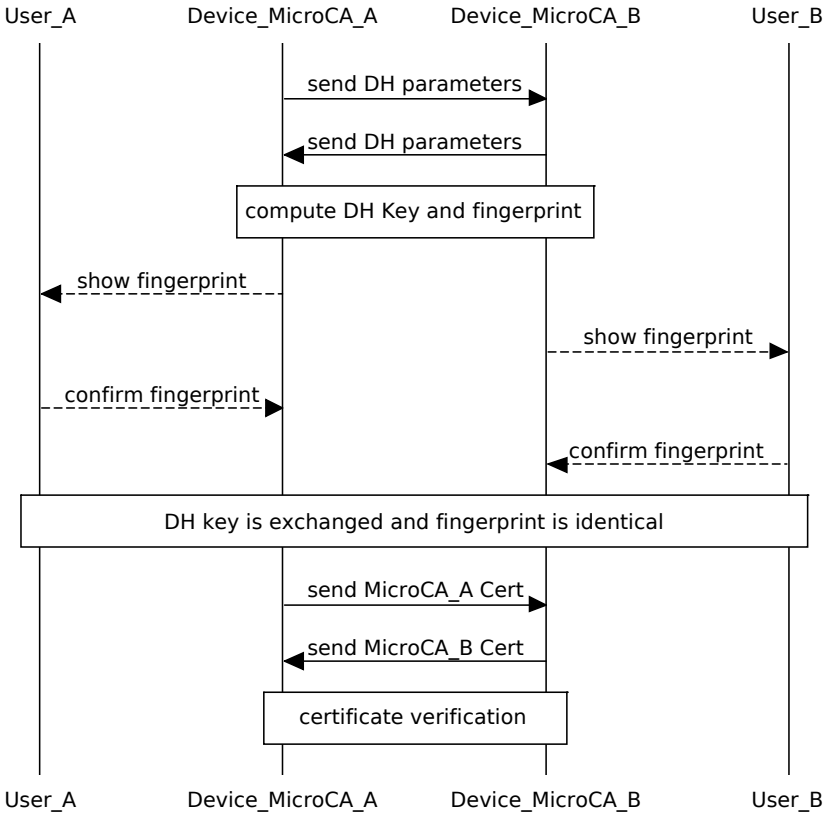

Fig. 7.4: Sequence diagram of the direct pairing mechanism

Therefore, we propose to use already established relationships of social networking sites such as facebook to initiate the pairing mechanism. We argue that the profile of a user with all his social relationships, pictures and posts is much easier to verify (and harder to fake) than a plain email address. In order to further validate the authenticity of keys, we propose to follow the Web of Trust approach where users cross-sign the keys of others. Once a decision about the authenticity has to be made, a user ranks all signatures and decides whether he wants to trust it or not.

Additional Entities for the Remote Pairing Process

In addition to the paring service as depicted in figure 7.1 our remote pairing protocol relies on two additional entities: A **Locator Storage** (LS) and a **Rating Storage** (RS). The LS stores the current network address (e.g. IP address and port) of the remote pairing service running on the MicroCA that is responsible for the ID. A user may publish a link to the LS on his personal social networking profile, which allows visitors to directly communicate with the networks pairing service.

The RS holds ratings from other users about the trustworthiness of the binding of the identity to a public key. The result of querying the rating storage helps to be sure about the authenticity of a remote identity to a certain degree and is suitable for granting access to many standard services. Ratings are generated and added to the storage once two users meet personally and perform the direct pairing as described above.

Remote Pairing Protocol

The actual protocol for the remote pairing process between two users A and B and their corresponding MicroCAs is shown in figure 7.5. User A queries the Locator Storage and receives the current locator for a given ID of user B. For example, user A may have visited the facebook profile of user B and by clicking on a published link his pairing service automatically connects to the remote pairing service.

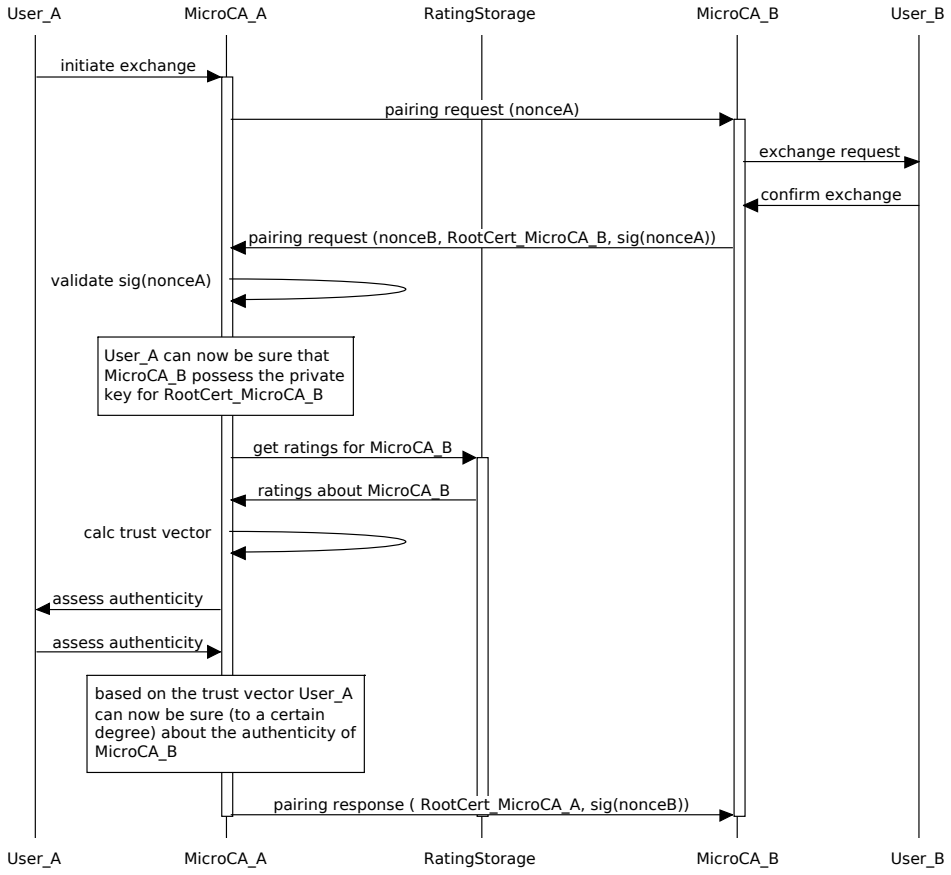

Fig. 7.5: Sequence diagram of the remote pairing mechanism

The pairing service of A first sends a pairing request that contains a random number ($nonceA$). This number is signed by B in order to prove that B actually possesses the private key for the certificate. B answers the pairing request by another pairing request that contains the signature ($sig(nonceA)$), its certificate, as well as another random number ($nonceB$). After validating the signature, user A queries the Rating Storage for ratings about user B. Ratings are signed by other users and dependent on the trust into them, user A is able to calculate his personal trust vector. Based on this vector user A decides about the trust in B's MicroCA.

7.6 Authorization

While authentication deals with the validation of the authenticity of an entity, authorization describes the access control to specific resources. For example, a successfully authenticated entity such as an employee may not be authorized to access the companies database. Therefore, it is essential for our security architecture to allow inexperienced

users to define and maintain fine-grained access rights to resources in the network. This section briefly explains relevant entities of our authorization approach and describes in section 7.6.1 how legacy services are able to use it. A detailed example is then given in section 8.2.

IETF RFC 2754 describes "a framework for policy-based admission control" [166]. The authors introduce an architecture that describes two essential entities for a policy-based management: a Policy Enforcement Point (PEP) and a Policy Decision Point (PDP). The PEP runs on the policy-aware node and enforces the policy decision that is made by a centralized PDP. In our architecture the Policy Decision Point is part of the MicroCA (see figure 7.1) and uses policies stored in the Entity Directory in order to make decisions. The owner of the network specifies polices via a user-friendly (web-based) interface (the Policy Manager) before storing them to the Entity Directory. As one instance of a real policy framework we use the eXtensible Access Control Markup Language XACML [97] to define and store policies, which are transformed automatically between the Entity Directory and the XACML policy database. More details on XACML are described in section 8.2. The Policy Enforcement Point may be part of the service itself, but we decided to also implement one generic PEP instance in the MicroCA as well. This allows connecting legacy services that do not implement a PEP by only defining a tiny XML-based protocol as described in the next section. If an application already provides a PEP, it directly connects to the PDP to query the policy database (see figure 7.1).

7.6.1 SSL/TLS interception for legacy Services

To connect existing services to our architecture we require a service to present a valid identity for authentication. Once authenticated, our unique cryptographical identities as presented in section 7.4.1 can then also be used for authorization. One of the most common protocols for securing services in the Internet is SSL/TLS [34]. SSL/TLS provides a handshake protocol that allows to mutually authenticate two hosts using certificates. Once authenticated, the client is able to access the service and it is not possible to also consider authorization policies. The goal of our security infrastructure is to provide authentication and authorization for services in the network. Thus, we extended the SSL/TLS handshake protocol and defined a XML-based interface (see listing below) to query a PEP within the handshake. The PEP then transforms this query to the policy specific protocol (e.g. a XACML request) and replies back with its decision (permit or deny). Figure 7.6 depicts the interception of the SSL/TLS handshake. First, the client establishes the connection sending a *Client Hello* and requesting the service to provide its certificate. The *ServerHello* contains the certificate and a request for the client to also provide its certificate (mutual authentication). Since the certificates were issued by either the same MicroCA or by a trusted remote MicroCA (after the pairing process) both parties are able to verify them. However, after verifying the clients certificate, the service creates a request and asks the PEP if the client is authorized to access the service. Only if the certificate is actually valid and the PEP answers with a permit message, the certificate is declared valid for the SSL/TLS session and the handshake proceeds. If the PEP answers with a denied message, the certificate is declared invalid and the handshake is terminated. This mechanism adds authorization to many services that only provide authentication today.

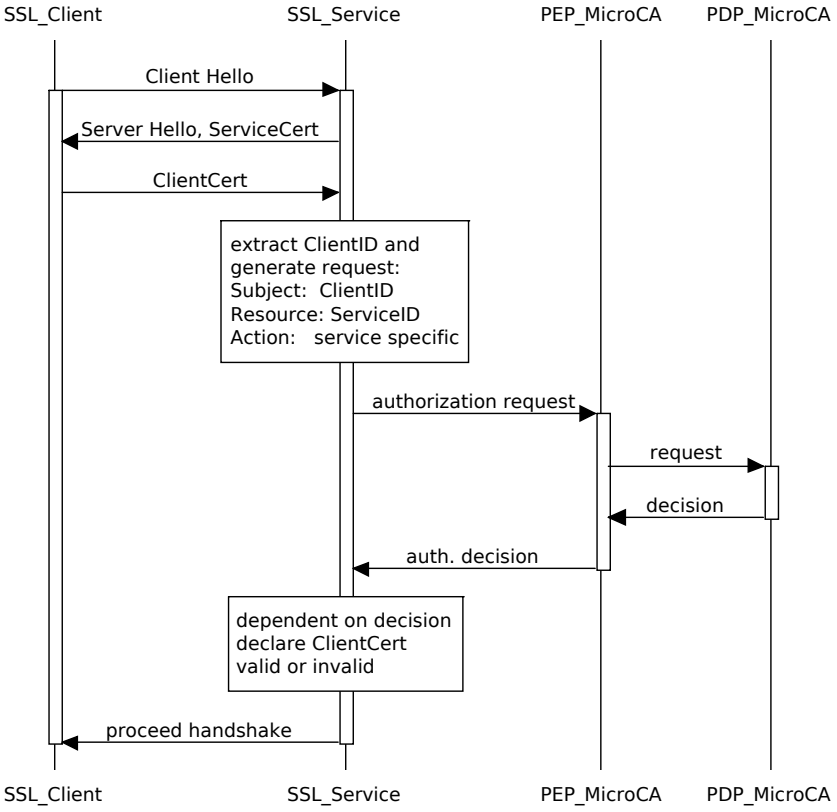

Fig. 7.6: Simplified SSL/TLS handshake interception

Subject:	ID_Client :	Who wants to access the service
Resource:	$ID\_Service$:	ID of the service
Action:	String: service specific	Service specific action (e.g. Lights On)

As a proof of concept we extended two SSL/TLS-based services with authorization: The RADIUS protocol as described in the next section, as well as DPWS, our reference example that will be described in the following chapter.

7.7 Implementation

We implemented a first instance of our security architecture to prove the applicability to unmanaged networks. For the Device Registration Service we chose to utilize the virtualization capability of WLAN access points in order to create two separate WLANs: an unprotected one (to prevent man-in-the-middle attacks, a pre-shared key (PSK) should be used) for connecting to the Device Registration Service and a protected one for the actual services. Figure 7.7 depicts the separation. The protected wireless network uses a RADIUS [125] server to authenticate clients based on their provided certificate. The RADIUS server (more specifically EAP-TLS [2]) was extended to connect it to our PEP as described above. This allows specifying fine-grained policies for network access.

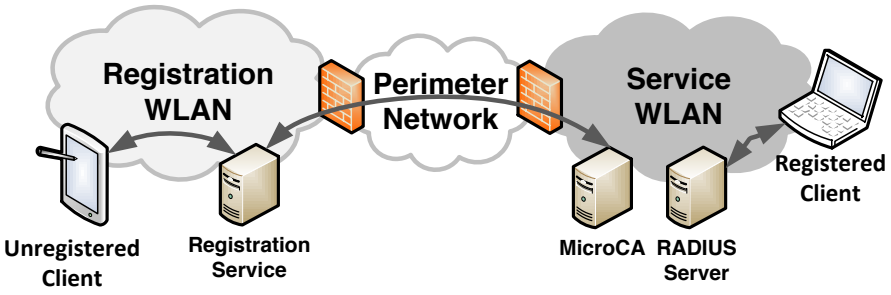

Fig. 7.7: WLAN implementation of the device registration process

The client side is implemented as a tiny application on Linux that provides a GUI for the registration process as described in section 7.4.2. We use Zero Configuration Networking (Zeroconf) to automatically discover the registration service. Thus, the client assigns itself an IPv4 link local address [19] and queries the network via Multicast DNS (mDNS) [21] for a registration service. We use the DNS-based service discovery (DNS-SD) [20] protocol with a new record called *_DevRegSrv_MicroCA._tcp*. The registration service replies back with its IP address and port and the user's device directly establishes a TCP connection to it. Now the actual registration protocol runs using an XML-based encoding. Finally, once a client has obtained a certificate from the MicroCA, it is able to access the service WLAN by connecting to the RADIUS server.

7.7.1 Hardware-based Security Extension

The private key of an entity, and in particular the one of a MicroCA, is a high priority target for attackers and therefore further security mechanisms are desirable. Large scale commercial CAs secure their private keys using very costly Hardware Security Modules (HSM), which is not a feasible solution for smaller networks. Smartcards store security tokens and perform cryptographic algorithms inside a closed environment. Because of their broad availability, they are a good choice for unmanaged networks. Finally, a Trusted Platform Module (TPM) [152] provides mechanisms to generate and securely store keying material. Additionally, TPM allows proving to a remote party that certain keys are stored in a way that they cannot leave the hardware container. This means, TPM protected keys are never exposed to the computer's main memory and cannot be extracted by software attacks. It would be easily possible to extend the above pairing mechanisms in a way that trust is only established into networks that are able to prove that they protect their keys via a TPM.

7.7.2 Integration

To integrate the presented services into one box, the MicroCA, we implemented a reference architecture based on host and network virtualization on an Intel Atom based system. The protection and separation of the Entity Directory and the Device Registration Service is reached by introducing a virtual perimeter network. The connection between the registration and service network is necessary, because newly registered clients have to obtain a certificate from the certification service.

Figure 7.8 shows the simplified architecture of the reference implementation. We used XEN[2] and a number of unprivileged domains for hosting the individual services.

[2] http://xen.org/

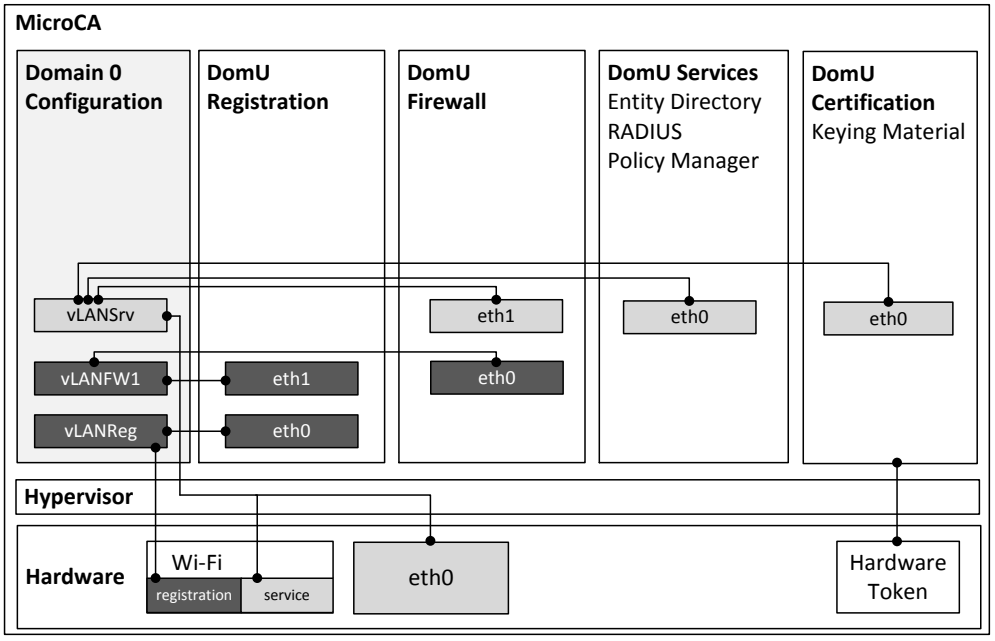

Fig. 7.8: Architecture of our implementation

Dom0 bridges the different networks and enables the communication between them. The separation of the registration and service network is done within the firewall DomU by letting an *iptables* process filter all traffic between eth0 and eth1.

7.8 Possible Attacks and Recommendations

Compared to the state of the art our proposed protocols and integrated architecture can be seen as an important step towards more security in unmanaged networks. However, there are a number of attack vectors to be considered when actually deploying our solutions.

When registering new devices to the MicroCA (Device Registration Process) we have to make sure to actually connect to the correct MicroCA. A potential attacker may try to force the device to first register to a fake MicroCA, which then acts as a man-in-the-middle between the device and the actual MicroCA. To prevent this attack, an authenticated Diffie-Hellman key exchange between the user's device and the MicroCA should be performed. If we assume physical access to the MicroCA, the DH fingerprint could be shown on a small display on the MicroCA itself. Here, user-friendly ways of showing ASCII images instead of fingerprints are possible (as proposed in [115] and implemented e.g. in *openssh*[3]). The user then compares the fingerprint as shown on his device with the fingerprint shown on the MicroCA and can be sure that no man-in-the-middle is involved. Another option is to use NFC for the paring process and rely on the fact that the radio only works within a distance of less than 10cm. An attacker would have to manipulate the MicroCA to run a man-in-the-middle attack. However, possible future attacks on NFC may still justify the use of the authenticated DH. When using WLAN for pairing, a pre-shared key for the registration network should also prevent from such attacks. However, the longer (and thus more secure) the passphrase, the worse the usability. Thus, we highly recommend the use of an authenticated DH key

[3] http://www.openssh.org/

exchange and the comparison of the fingerprints independent from the actual pairing technology.

The same problem (fake MicroCA) applies to the remote pairing process. Our system uses social networks as a trust anchor for the binding between an identity and its public key. An attacker may be able to fake a social network profile (with all its social relationships) or hack into the correct profile and change the public key (or the link to the LS), resulting in a man-in-the-middle attack. Thus, an authenticated DH using a sidechannel (e.g. email, phone) for verifying fingerprints is also highly desirable. Additionally, our trustvector describes the level of trust for an identity and it should be noted that a remote pairing is most likely not as trustworthy as a direct (face-to-face) pairing. We therefore recommend that every remote pairing is verified at some time by a direct pairing and that the authorization policy should take the trustworthiness of an identity into account.

7.9 Summary and Key Findings

This chapter described the insights of our secure service infrastructure for unmanaged networks and presented building blocks that help to considerably improve the security of unmanaged networks. We are convinced that future scenarios for unmanaged networks will benefit from security mechanisms that are common in today's enterprise networks and show that an adaption to unmanaged networks is possible. This contribution answers question Q3 according to section 1.1.

We identified the combination of a local Certification Authority, the MicroCA, and the Web of Trust approach as suitable for unmanaged networks, because it allows 1) easily managing all devices, users and services within one network and 2) establishing trust into a complete "key-family" (all keys signed by the MicroCA) rather than in only one key. Our system assists inexperienced users in maintaining their own PKI and hides the technical complexity from them. Trust establishment mechanisms for a direct pairing and a remote pairing via social networks were presented. We implemented a first instance of the MicroCA and showed how a small device is able to host all needed services in a secure way.

Key Findings of this chapter
(and contributions according to section 1.2):

- **Inexperienced users are not able and/or willing to maintain enterprise grade security mechanisms, although they would benefit from them.**

C7.1a **The presented MicroCA architecture provides almost zero-configuration plug and play security to unmanaged networks.**

C7.1b **The Device Registration process describes an easy-to-use mechanism for issuing public key certificates to devices and users.**

C7.1c **By using already established trust relationships of social networks in combination with a Web of Trust like rating system, it is possible to establish a certain level of trust into a previously unknown identity.**

C7.1d **Our infrastructure allows providing authorization to legacy services through SSL/TLS interception.**

8. NEW MIDDLEBOX SERVICES

8.1 Introduction

In the previous chapters we have shown why middleboxes have been introduced to the network, what their impact is and which problems they impose. This chapter focuses on problems that can be solved by new middlebox services. First, section 8.2 presents a reference example for our security infrastructure showing how services can benefit from our security framework. We equip the Devices Profile for Web Services (DPWS) [42] with the authorization framework XACML in order to secure the access to specific services as defined by an administrator of the network. Additionally, we develop a middlebox that allows discovering and using services of remote trusted networks. Section 8.3 then targets privacy problems that have been introduced with the availability of IPv6. We develop a middlebox service that adds privacy to IPv6-based networks without relying on the host to support the IPv6 Privacy Extensions [109]. Additionally, the administrator is able to define policies for services to allow the reachability via fixed addresses. Connections to protected services are obfuscated by a checksum neutral translation mechanism as defined in [159]. Finally, section 8.4 presents an approach for combining host and network virtualization providing a great flexibility for interconnecting different networks and for isolating concurrent services.

8.2 A Middlebox for securing DPWS

Unmanaged networks and inexperienced users call for services that follow the plug and play paradigm. One of the most widespread plug and play protocols in unmanaged networks is UPnP [50]. UPnP allows a seamless discovery and usage of services for entertainment, data sharing and communication in general. However, UPnP offers no security mechanisms and as networks grow, protecting the privacy of data and services becomes more and more important. UPnP is not only vulnerable to attacks coming from the internal network, [56] shows that many UPnP-enabled routers also enable UPnP on the WAN interface. Several suggestions for securing UPnP [45] and for UPnP remote access [110] have been made, but none of them provide an integrated solution suitable for inexperienced users. This is mainly because providing security features in a way that non-experts can benefit from them is not a trivial task. Our security infrastructure as described in chapter 7 can be used as a basis for securing services in unmanaged networks.

As a possible successor of UPnP, the Devices Profile for Web Services (DPWS) [42] is built upon the OASIS Web Services (WS) stack and implements basic security components (access to a service via SSL/TLS) by default. However, DPWS offers no possibility to define fine-grained policies for controlling access to certain functions only (DPWS actions). For example, browsing subfolders of a media collection cannot be restricted to a user or a user group. Browsing can be either allowed (if the client provides a valid certificate) or denied.

This section illustrates how DPWS can be connected to our MicroCA infrastructure and extended by the policy framework XACML [97] to reach two improvements over the state of the art: First, we show how to restrict the access to DPWS actions to authorized hosts only by intercepting the SSL/TLS handshake as proposed in section 7.6.1. Secondly, we present a new middlebox service that allows using DPWS across multiple networks. Here, we in particular have to make sure to limit the discovery of available services to authorized hosts.

8.2.1 Technology Overview

DPWS

The Devices Profile for Web Services (DPWS) distinguishes between a hosting service representing a device (or a server) and the actual hosted service (see section 6.5.1). A hosted service and its offered actions (e.g. the hosted service might be a light switch that offers two actions: lights on and lights off) can be protected from illegitimate access by providing the Web Services Description Language (WSDL) [23] over a mutually authenticated HTTPS connection. However, once a client has been authenticated, it is allowed to execute all the actions the service implements. For many scenarios (e.g. guests in a home network) it is important to restrict the access of a service at the action level instead of at the service level only.

The eXtensible Access Control Markup Language (XACML)

The eXtensible Access Control Markup Language (XACML) is a language for describing authorization and privacy policies and was standardized by the OASIS consortium [97].

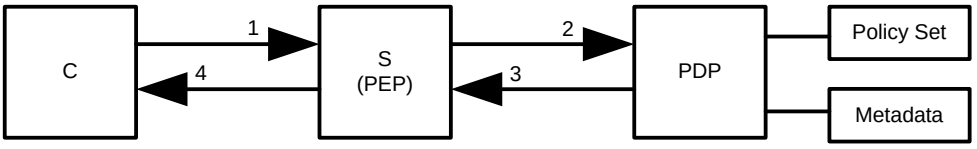

Fig. 8.1: Main XACML entities

The XACML architecture consists of three main entities as depicted in figure 8.1: a client application C that desires to access a service S (the Policy Enforcement Point (PEP)) and a Policy Decision Point (PDP). Upon a service request of C (1), S creates a XACML query that contains the ID of C (*Subject*), the ID of S (*Resource*) and an *Action* that should be executed on S (e.g. read or write) and sends this query to the PDP (2). The PDP evaluates the query using a pre-defined policy set and additional meta information and sends the decision back to S (3). The answer to the initial query (permit or deny) is then dependent on the PDP's decision (4).

A XACML policy set as depicted in figure 8.2 consists of at least one policy. The PDP first evaluates which policy to use by matching the *target* information (subject, resource and action) of the policies. After finding the right policy, the PDP evaluates which rule to use. A rule specifies under which conditions a request should be allowed or denied.

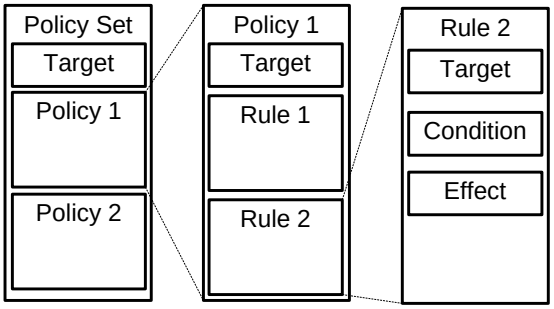

Fig. 8.2: XACML policy set

8.2.2 Scenarios and Contributions

Our application example focuses on two scenarios. First, only one network that is equipped with a number of DPWS services is considered. Members of this network should be able to discover DPWS services and query DPWS actions. With the state of the art and standard DPWS, service access (to all actions) is granted if the requester provides a valid certificate. Our solution allows the definition of fine-grained policies to restrict the access to certain actions dependent on the policies for the requesting entity.

The second scenario covers the access to DPWS services from a remote network. This requires the cooperation of at least two networks and a possibility to restrict access not only to certain actions, but also to the discovery mechanism of DPWS. Therefore, a secure middlebox that allows the tunneling of multicast discovery messages across the Internet is also part of our solution.

8.2.3 Approach

Figure 8.3 shows the main entities that are involved in the considered scenarios. In the first step a DPWS client sends a probe multicast message to its own network searching for available DPWS servers. In addition to local DPWS services, the middlebox (MB) (here implemented as a proxy) responsible for the client's network receives the messages and establishes a secure tunnel to the according remote trusted middlebox.

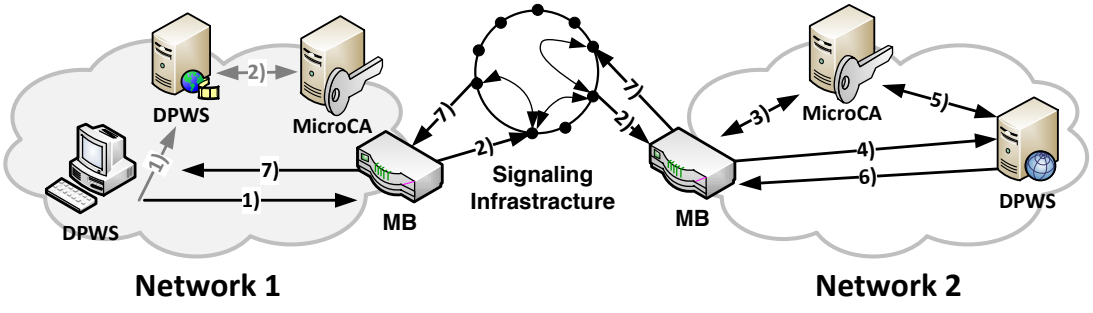

Fig. 8.3: Entities for our two scenarios

Once the remote middlebox receives the probe message, it checks with the local Policy Decision Point (PDP) located at the MicroCA if there is any policy matching the incoming packet. This means, the middlebox enforces the policies defined for the network (policy enforcement point (PEP)). Finally the probe message is multicasted into the local network 2 and the probe match message can be sent back to network 1.

Equipping DPWS with XACML

After the discovery process the client may eventually want to access a DPWS action as defined in the WSDL file provided over a HTTPS connection. In order to restrict the access to certain actions (e.g. lights on/off), the DPWS hosting service is extended by the SSL/TLS interception mechanism as described in section 7.6. This means, whenever a client asks for a connection to an action, the service not only verifies the provided certificate, but also asks the PEP and PDP in the network (located at the MicroCA) if the client is allowed to access the action. Once the certificate has been verified, the hosting service extracts the public key from the certificate and creates the client ID by calculating *hash(ClientPubKey)* according to 7.4. The globally unique ID *ClientID.MicroCAID* is then mapped to the subject field of the (XACML) request. The resource field is the ID of the hosted service and the action field is the DPWS action the client is requesting. The PDP evaluates the request with the help of additional metadata, which are XACML attributes based on the clientID or its MicroCAID, and sends back the response to the PEP. This provides the flexibility to control the access levels for a remote network, as well as for a specific foreign client. Section 8.2.4 shows how templates for these policies can be auto-generated for each hosted service.

8.2.4 DPWS service usage across Networks

DPWS, as well as UPnP, is restricted to only one broadcast domain because it uses IP multicast for discovery. For UPnP, which provides no security at all, a remote usage is not desirable. For DPWS we might also want to allow the discovery of devices across domains, because it is possible to restrict the access on the service itself by using certificates. There have been discussions and first implementations of discovery proxies for DPWS[1], however a complete system providing discovery, signaling, authentication, authorization and finally remote service usage does not exist.

DPWS Middlebox Proxy for interconnecting Networks

To enable the remote discovery of DPWS devices we implemented a DPWS interconnection proxy that forwards DPWS discovery messages to remote trusted networks. The (TCP) connection to the service itself is then established directly from the DPWS client to the server. This is because these messages may be encrypted (SSL/TLS) and cannot be handled by the proxy. To allow the traversal of possible middleboxes on the path we integrated our middlebox traversal framework as presented in chapter 6.

Whenever a device queries the network asking for a service (DPWS probe) the DPWS proxy gets this packet (step (1) in figure 8.3), analyzes it and passes it to the remote DPWS proxy (step (2)), which sends the multicast packet out to its own network. The remote proxy then acts as a client on behalf of the requesting device and therefore gets a reply (probe match) back from the actual service (step (6)) located in the remote network (the complete DPWS discovery phase is depicted in figure 6.11 of chapter 6). This requires the proxies to maintain state and to map between the local network and the identifier of the remote proxy. In our implementation the client proxy also maintains a database where users can configure to which remote homes a probe message should be forwarded. For example, when searching for a media server the client proxy may forward this query to all trusted networks.

[1] http://msdn.microsoft.com/en-us/library/system.servicemodel.discovery.discoveryproxy.aspx

Remote networks are identified by globally unique identifiers (e.g. as described in chapter 7) and a signaling infrastructure (e.g. a peer-to-peer network) is used to resolve IP addresses of remote middleboxes. Finally, the proxies establish a secure tunnel (e.g. SSL/TLS) and authenticate each other using service certificates issued by the MicroCAs (a trust establishment between the involved parties is necessary to verify the service certificates, see section 7.5). Therefore, all DPWS servers can be sure that requests only come from trusted remote networks. More security considerations are discussed in section 8.2.5.

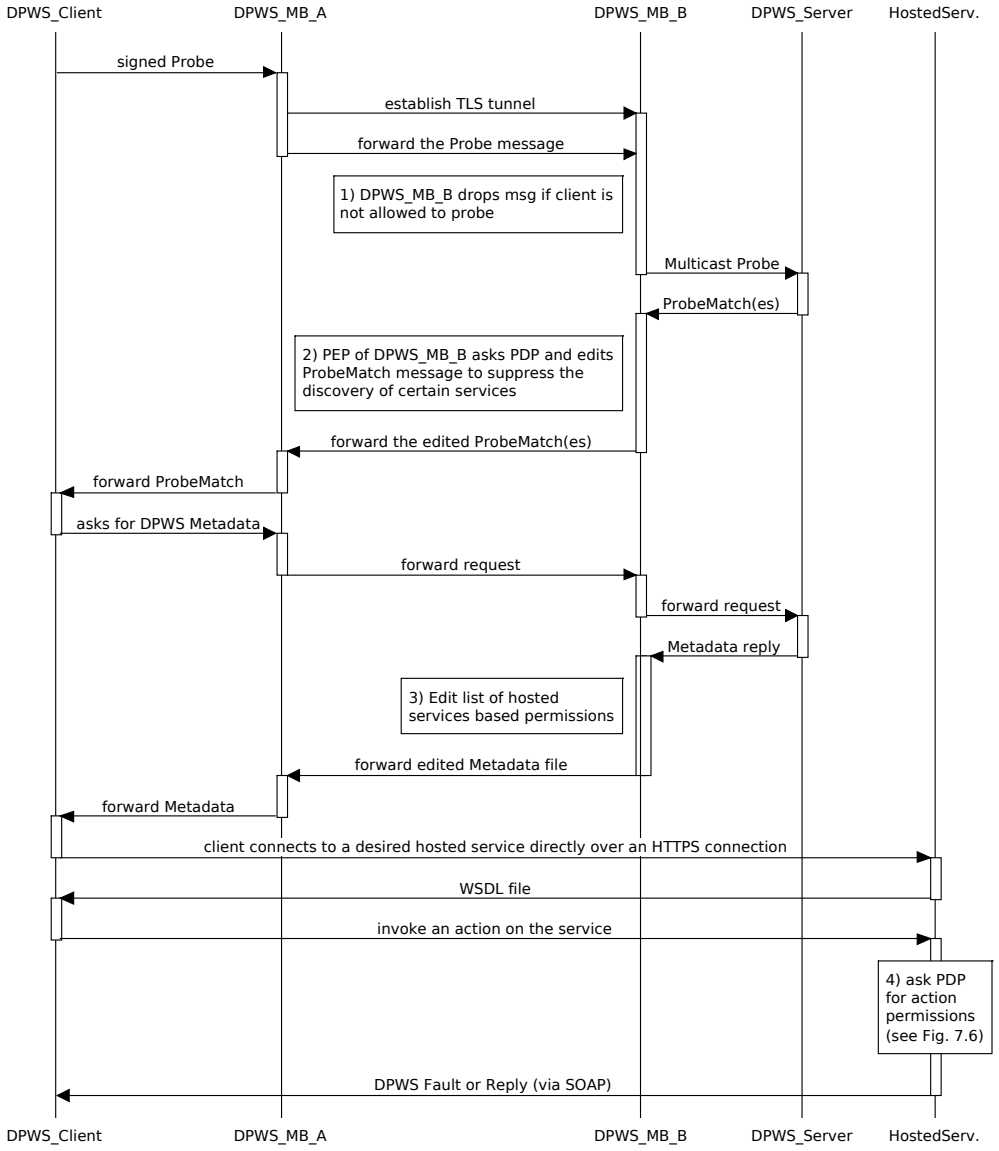

Fig. 8.4: Protecting DPWS using a middlebox proxy

DPWS Firewall

With the proxy described above the system is now able to discover and use services that are located in trusted remote networks. The remaining questions now are a) how to suppress the forwarding of incoming probe messages and b) how to make sure

that a client is only able to discover the services it is allowed to. To use XACML for this purpose, we equip the proxies with PEP functionality as described in section 7.6.1. Figure 8.4 shows the individual steps that are necessary until a direct HTTPS connection to the DPWS action URL can be established.

After receiving a signed probe message (containing a query for a certain DPWS device) from an internal device (*DPWS_Client* according to figure 8.4), the middlebox of network A (*DPWS_MB_A*) extracts the public key from it, calculates the clientID (*hash(pubKey).MicroCAID*) and asks the network's Policy Decision Point (PDP) if the client is allowed to send discovery messages to remote networks. If so, the probe message is forwarded to the remote network and middlebox *DPWS_MB_B* decides if it accepts it ((1) in figure 8.4). The resulting probe match message as generated by the DPWS server, as well as outgoing metadata messages contain a list of hosted services, which may be edited by the middlebox if the requesting client is only allowed to discover a subset of them (steps 2 and 3). Finally, the client invokes an action on the DPWS hosted service, which enforces the policy by querying the PDP as described above and depicted in figure 7.6 of chapter 7.

Policy Creation

When aiming at protecting the discovery of services (suppress certain services for unauthorized clients already during the discovery phase), as well as the access to certain DPWS actions, two types of policies are needed: The first policy set defines which clients are allowed to discover which services, whereas the second policy defines which client (client ID mapped to subject field of XACML) is allowed to access an action (DPWS action mapped to action field of XACML) of the resource (service ID mapped to the resource field of XACML). Policies for actions can be automatically derived from the appropriate WSDL file of the DPWS hosted service (see figure 8.1). The *Resources* field in the target section of an XACML policy is mapped to the ID of a service and the *Actions* field contains multiple XACML actions; one for each DPWS action (port-type) as defined in the service WSDL file. The *Subjects* field in the target section and the *Rules* in the policy are kept blank and can be edited later by the administrator, if and when required. A default *Rule* can be added as per the network configuration to permit/deny all requests.

```
<Policy>
  <Target>
    <Subjects/>
    <Resources>
      <Resource> <AttributeValue>MediaServer</AttributeValue> </Resource>
    </Resources>
    <Actions>
      <AttributeValue>GetDirectoryContent</AttributeValue>
      <AttributeValue>PlayFile</AttributeValue>
      <AttributeValue>Control</AttributeValue>
    </Actions>
  </Target>
  <Rule Effect="Permit" RuleId="Default">
    <Target/>
    <Condition> <Apply/> </Condition>
  </Rule>
</Policy>
```

List. 8.1: Skeleton of a XACML policy

8.2.5 Security Discussion

Our system aims at enabling authentication and authorization for service discovery and for the access to DPWS actions. When discussing the security properties of our system, we differentiate between a) access to local services within the network and b) access across networks to remote services.

Attacker Model

Our attacker model follows the Dolev-Yao model [40], where an attacker is able to eavesdrop, send, receive, change, replay and fake messages. We consider attackers located in the public Internet, as well as attackers located within the trusted networks. However, clients that possess valid certificates signed by the local or a trusted remote MicroCA behave protocol conform and attackers are not able to steal private keys from legitimate clients.

Access to local services

When considering a scenario with only one local network (DPWS broadcast/multicast domain, thus no middleboxes), our system behaves like standard WS-Discovery. All devices within the local network (legitimate clients, as well as attackers) are able to send DPWS probe messages via IP multicast and are therefore able to discover all services hosted by DPWS servers within the local network. If probe messages are signed by clients, the server is not only able to verify their integrity, but also able to extract the client's identity. The client's identity can then be used by the DPWS server to query our Policy Decision Point and to determine if certain services should be excluded from being discovered. This is an important improvement compared to standard DPWS and excludes attackers that do not possess valid certificate from discovering services. An attacker might try to run a denial of service attack by letting the DPWS server check the signatures of many invalid messages. This can be prevented by first checking the validity of the provided key before actually checking the signature.

Since the actual access to DPWS services is provided via HTTPS connections, clients that do not possess valid certificates are not able to access them. Furthermore, since our solution also adds authorization, accessing a services requires a valid certificate, as well as an authorization policy, an additional improvement compared to the state of the art.

Remote access via proxy

For service discovery and usage between trusted networks our secure SSL/TLS tunnel prevents attacks coming from the public Internet. Additionally, we only forward discovery requests originating from trusted networks, which excludes arbitrary attackers located in untrusted remote networks. Attackers within trusted networks are able to send discovery messages, allowing them to discover all services within the local and the remote network. Again, we suggest that inter domain discovery messages (probe messages) are signed using the compact signature format as defined in the Web Services Security standard [107]. This enables a network to authenticate remote clients and determine whether a) the client is allowed to send discovery messages into the own network and b) to filter probe matches according to some policy in order to hide services to clients from remote networks. For unsigned discovery messages the DPWS

proxy is still able to determine the (trusted) origin of the request (on a per network basis) and filter probe match messages accordingly.

As the WS-signature operates on the application layer and only signs the actual DPWS message and not the underlying transport layer, an attacker in the remote network might eavesdrop a legitimate discovery message and replay it. The only protection against this is to log message IDs and discard duplicates. Filtering of duplicate messages can be done locally by the network, but could lead to additional load (possible DoS attack). As the remote network is generally trusted, we propose to host this message filter in the remote network. Additionally, the local network can filter messages using a short sliding window of message IDs that will limit resource consumption.

8.2.6 Evaluation

Table 8.1 shows our achievements and compares them to the state of the art. The combination of our authorization framework and an SSL/TLS-based service such as DPWS allows defining fine-grained policies not only for accessing local services, but also services in remote (trusted) networks.

	UPnP	**Standard DPWS**	**DPWS plus MicroCA**
local access to actions	– unrestricted no security	(+) unrestricted valid cert. is sufficient	+ restricted on a per client basis
local discovery	– unrestricted no security	– unrestricted no security	(+) restricted for signed probes
remote discovery	– not possible	– not possible	+ restricted per client or remote MicroCA
remote access	– not possible	– not possible	+ restricted per client

Tab. 8.1: Comparison to the state of the art

To prove that our system works as expected, we implemented a DPWS video streaming solution, which is described in [101]. Video/audio streaming can be seen as an example application representing unmanaged networks (especially home networks) and as bandwidth grows, multi domain scenarios are most likely to appear. This means the streaming between multiple home networks should work as automatically and self-configuring as the discovery and streaming within one home.

We then measured the processing time at the middlebox between receiving a probe message from a remote network until forwarding the probe message to the local network. This includes the processing of the messages, as well as the XACML query. The DPWS server was not involved, because it simply implements a second PEP querying the same PDP as the proxy. Figure 8.5 shows the results. The prototype was run on a standard 2010 dual-core linux machine and the PDP implementation of SUN[2] was used.

[2] http://sunxacml.sourceforge.net

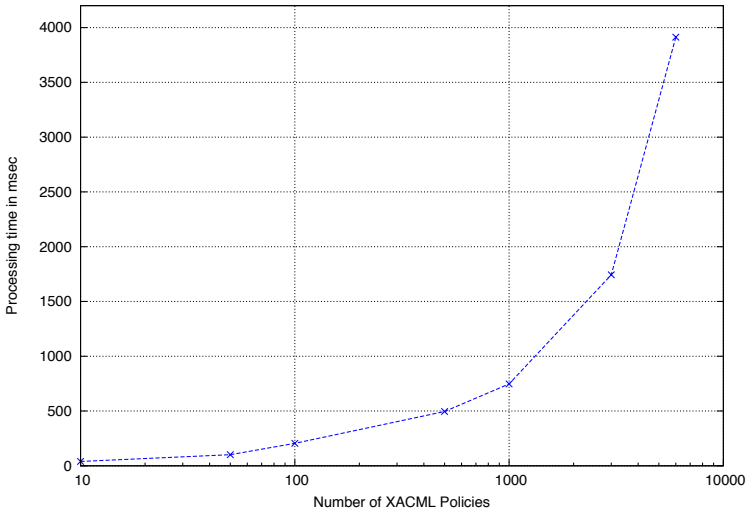

Fig. 8.5: The processing time in milliseconds dependent on the number of policies

While the processing time of the middlebox itself was only approximately $1ms$, the XACML lookup is dependent on the number of policies that are maintained at the PDP. We differentiate between policies needed for discovery and policies needed to secure the actions of a service. While every hosted service in the local network may get its own policy for discovery, the handling of policies for services is application specific (dependent on the number of actions the service implements). For example, a audio/video streaming service may hold a policy for each media file or it may hold only one policy for a complete media collection. Furthermore, we propose to run multiple PDP instances in parallel, one for discovery and one for the applications built on top of DPWS. Thus, the discovery process cannot be blocked by a large number of policies needed for a specific service (e.g. for accessing a media collection). Finally, for less complex scenarios only requiring a few policies (e.g. one policy for a media collection), text-based policy files might be sufficient to avoid the overhead of parsing XML. It might also be desirable to evaluate the performance of other PDP implementations, which claim to perform better than the SUN implementation.

8.2.7 Summary

In many unmanaged networks the only service that offers security is the access to the wireless network. Popular service frameworks such as UPnP offer no security features and impose many problems. In this section we showed how arbitrary SSL/TLS-based services can be secured by our MicroCA approach as presented in chapter 7. As a reference example we extended the Devices Profile for Web Services framework by the ability to not only deliver its content via a secure connection, but also to restrict the access to DPWS actions to authorized clients. Additionally, we developed a middlebox service that allows the secure discovery of remote DPWS services and the filtering of services during the discovery process. We showed how to use the security framework XACML for authorization and for defining fine-grained policies for discovery and service access.

8.3 PrivMid6: A Privacy Preserving Middlebox for IPv6

The large address space of IPv6 (128bit) removes the need for Network Address Translation and allows allocating globally unique IP addresses to individual devices. While this re-establishment of the end-to-end argument on the one hand solves many of the issues of IPv4, it also introduces new problems. One of the most profitable business models in today's Internet is to sell customized advertisements. Thus, companies such as Google and facebook try to collect as much information as possible about user behavior and aim to track users across multiple websites. On the technical side, this requires to identify individual users based on unique tokens. With dynamic public IP addresses (as allocated by ISPs for IPv4), user tracking based on the network layer is rather difficult since middleboxes such as NAT and Large Scale NAT obfuscate the actual private IP address of the client by translating it to a global one. Thus, a complete network only uses a few (in many cases only one) global addresses. With globally unique IPv6 addresses tracking on the network layer is easily possible.

IPv6 addresses are usually created on the host following the stateless auto-configuration protocol as described in [150]. The least significant 64bits of an IP address are hereby derived from the unique layer 2 address using the EUI-64 identifier [74]. Thus, the device identifier of a public IPv6 address only depends on the host and is therefore easily identifiable since it does not change even when changing the global prefix. Another problem is related to enterprise networks. NAT for IPv4 is often used by larger companies to hide their topology from the outside world, thus introducing privacy for their topology. Instead of using their available IPv4 address space, only a small number of addresses are visible to the public. Without topology hiding, internal addresses are exposed to the public, which eases scanning and eventually attacking internal hosts. Additionally, Network Address Translation provides address independency when allocating addresses to internal hosts. As a result, only addresses of the border router have to be changed when migrating to another ISP.

To overcome the privacy problems of a unique device identifier, the state of the art proposes to randomize IPv6 addresses on the host following the IPv6 Privacy Extensions [109] standard. However, if privacy extensions are not available (e.g. on mobile phones) or enabled on the host, privacy can not be provided. Additionally, the state of the art only randomizes the device identifier. For smaller networks that only need a few subnets there is many room for improvement by also randomizing additional bits. Finally, the randomized IPv6 address as generated by the original privacy extensions is not dependent on the prefix. Thus, if a host switches to a different network, the (randomized) device identifier remains the same and the host is still identifiable.

This section proposes a new middlebox service that is introduced as a gateway to IPv6-based networks and takes care of randomizing IPv6 addresses. The middlebox is designed as a hybrid solution: users are able to specify policies describing which services should be obfuscated and which services should still be globally reachable. Obfuscation is provided by translating between local and randomized public addresses. As the translation of layer 3 addresses also affects the layer 4 checksum, the external address has to be checksum-neutral to the internal one. Finally, our middlebox not only provides privacy, but also addresses shortcomings of the IPv6 privacy extensions.

The following sections present our approach, which was initially developed in [10], in more detail. After covering related work in section 8.3.1 and our scenarios in section 8.3.2, section 8.3.3 presents our requirements. Section 8.3.4 then describes our approach and the design decision we have made. Finally we evaluate our middlebox service in section 8.3.5 and summarize our findings in section 8.3.6.

8.3.1 Related Work

Many protocols and applications exist that aim to anonymize IP-based traffic in a way that the receiver of a packet cannot identify the sender. Examples are Tor[3], I2P[4] and JAP[5]. Most of them are based on public key cryptography as initially proposed in [18] and use the onion routing approach as presented in [37]. Here, each message is encrypted multiple times and sent via a number of proxies. Each intermediate proxy then decrypts and removes one layer before sending it to the next hop. This ensures privacy and untraceability since only the last hop is able to see the plain message and tracking between intermediate hops is not possible. However, this approach introduces overhead and the performance depends on intermediate hops and especially on the exit-node. An analysis of the performance of Tor is presented in [38].

Other approaches do not aim for complete anonymity, but only care about profiles that are created based on the IP address. The most important solution in the state of the art providing privacy to IPv6 is called "Privacy Extensions for Stateless Address Autoconfiguration in IPv6" [109]. Instead of creating the 64bit device identifier based on the layer 2 address, the privacy extensions create random identifiers with limited lifetimes. Thus, IP addresses change over time and are not related to the static unique MAC address of the host. While many desktop operating systems enable privacy extensions by default, mobile operating systems such as Android and iOS either don't support them or require administrative privileges for activating them. Thus, the level of protection depends on the host, the user and the operating system.

With IPv6 privacy extensions only the last 64bit of the IPv6 address are randomized. A network with only a few devices can still be identified by considering the more or less static prefix that is assigned by the ISP. Providers such as Deutsche Telekom plan to assign /56 or even /48 prefixes to their customers, thus allowing to randomize additional bits. The IPv6 randomizer[6] as presented by the German magazine Heise[7] provides scripts for the router firmware OpenWRT[8] to change the announced subnet from time to time. The solution relies on activated privacy extensions on the hosts and only uses one subnet at a time, which still allows to correlate individual hosts belonging to one network.

In [9] the authors present weaknesses of the original privacy extension algorithm. IPv6 privacy Extensions generate their random IPv6 addresses based on a hashed random value and do not depend on the announced prefix. Thus, if only the prefix changes (e.g. due to a network change), the random address stays the same and a host is trackable across multiple networks. The authors therefore propose an alternative way of generating random interface identifiers by also considering the prefix when hashing the random number.

[3] https://www.torproject.org/
[4] http://www.i2p2.de/
[5] http://anon.inf.tu-dresden.de/
[6] https://bitbucket.org/reik/ipv6randomizer
[7] http://www.heise.de
[8] https://openwrt.org/

8.3.2 Scenarios

Our proposed middlebox service targets consumer networks, as well as enterprise networks and mobile operators that aim to provide privacy to their customers. For home networks and other unmanaged networks our middlebox service may replace today's home routers and may be used to connect a home network to the Internet. Existing routers still focus on IPv4 and only offer few IPv6 features, excluding privacy and topology hiding. Future home routers should cover these new issues that arise with the introduction of IPv6, especially once IPv6 becomes the default network layer protocol.

For enterprise networks our middlebox can be used in addition to the existing infrastructure providing randomization and translation of IPv6 addresses. Here, topology hiding and address independency are additional useful features that are provided by our solution. The same is true for ISPs (e.g. mobile operators) that want to offer privacy services (e.g. as a new business model) to their customers. Instead of randomizing IPv6 addresses on the customers device, our middlebox allows to track customers in the internal network (legal interception), while at the same time providing them privacy towards the public Internet.

8.3.3 Requirements

If available and configured, IPv6 privacy extensions randomize the 64bit host part of an IPv6 address. Since ISPs plan to allocate a /48 or /56 network to consumers and even larger prefixes to companies, there is further room for improvement. We therefore propose to also randomize the subnet bits and to also support hosts that do not implement the IPv6 privacy extensions. The following list presents the requirement of such a solution:

- **R1: Anonymization**: The middlebox service should provide the anonymization of a definable length. For example, a length of 0 means that the middlebox behaves like plain IPv6, a value of 64 emulates the IPv6 privacy extensions and 128 would randomize all bits. Thus, each network administrator can define the grade of anonymization based on the network prefix that is assigned by the ISP[9]. Additionally, other host-based privacy techniques may coexist without influencing the behavior of the system.
- **R2: Topology Hiding**: The middlebox should hide the topology of the internal network.
- **R3: Protocol Independent Translation**: The translation of IPv6 addresses (if necessary) should be done in a transport protocol friendly way and not affect TCP and UDP.
- **R4: Legal Interception**: Anonymity aims to prevent external entities from tracking user behavior based on IPv6 addresses. However, the operator of the middlebox service should always be able to reconstruct the mapping between the internal and external address.
- **R5: Allow Incoming Connections**: The advantage of having globally unique IPv6 addresses is the ability to establish connections between arbitrary devices. This allows to host services within edge networks and applications such as Voice over IP don't suffer from realm-specific addresses. Thus, our middlebox should also support to exclude certain addresses from being randomized.

[9] The maximum grade of the anonymization depends on the length of the assigned prefix.

8.3.4 Design of PrivMid6

The primary goal of the Privacy-aware Middlebox for IPv6 (PrivMid6) is to introduce privacy to IPv6-based networks by randomizing as many bits of the IPv6 address as possible in a way that does not affect overlaying transport layer protocols. PrivMid6 acts as the interface between two IPv6 networks and controls the flow of packets between two realms. Packet flows in PrivMid6 are controlled by a central policy allowing to define a trade-off between reachability and privacy. If the policy is set to reachability (either for individual addresses or for a subnet) packets are forwarded and not processed any further. For privacy critical packets PrivMid6 provides a checksum-neutral randomization of a configureable amount of bits (e.g. interface ID plus subnet). Since there is no relation between the internal and the external address, the middlebox maintains a state table to store the lifetime and the mapping between the internal and external address. Algorithm 8.1 shows the processing of outgoing packets (meaning packets that are sent from internal hosts to the public Internet) according to our processing model as defined in section 3.2. For incoming packets the reverse processing is applied, similar to NAT44 functionality as presented in section 3.2.

```
1  if found (policy := get_state(p)) then
2      case policy == forward
3          send(p, external);                          ▷ forward packet to output stage
4      case policy == translate
5          if found (pub := get_state(p)) then
6              p := translate_packet(p, pub);          ▷ translate packet immediately
7          else
8              pub := allocate_new_mapping(p);         ▷ a new mapping is needed
9              store_state(p);                         ▷ remember new mapping
10             p := translate_packet(p, pub);          ▷ translate packet
11         end if
12         send(p, external);
13     end case
14 end if
```

Alg. 8.1: $Proc(internal, p)$ for PrivMid6 outgoing packets

1. For each packet p arriving at the internal interface the policy database is queried and a processing thread is determined (line 1). Here the policy returns $forward$ or *translate*, but a firewall could also return a *drop* policy.
2. In case of a $forward$ policy the appropriate function in the outgoing stage is called (line 3).
3. In case of a *translate* policy PrivMid6 behaves like a stateful NAT device.
4. If the randomized IP address is still valid it is found in the database and passed to the translation function (lines 5 and 6).
5. If the randomized IP address is not valid anymore or the packet belongs to a new flow a new randomized IP address is generated in a checksum-neutral way (line 8) and stored to the mapping database (line 9).
6. Finally, the translated packet is passed to the outgoing stage by calling the *send* function (line 12).

Generation of Addresses

According to the processing model as shown in table 8.1 the middlebox has to generate randomized temporary addresses whenever a translation event occurs. There are two requirements for the new source address: first, it must not be possible to calculate the internal address for any given external address and second, the layer 4 checksum of the external address must match the layer 4 checksum of the internal one. Here, we follow the approach as presented by the IPv6-to-IPv6 Network Prefix Translator [159]. The actual generation of randomized addresses is based on the algorithm as presented in [9] and shown in figure 8.6. Here we assume that the ISP assigns a /48 network to its customers. If the prefix differs, the number of bits for the subnet will vary, but the method is still the same.

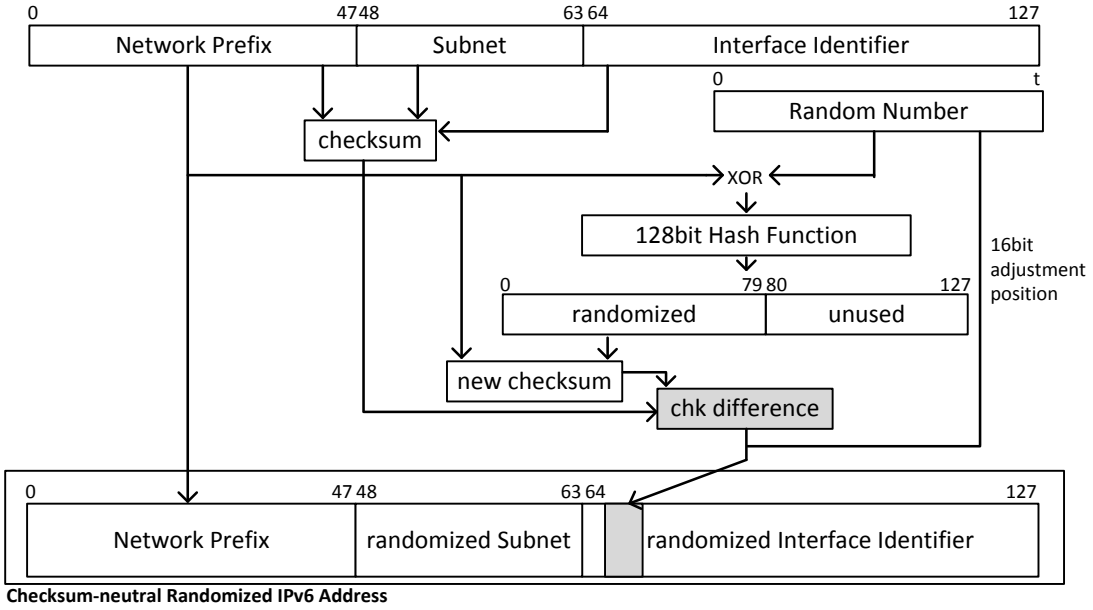

Fig. 8.6: Generation of addresses for PrivMid6

The initial IPv6 address on the host is configured by the stateless auto-configuration protocol as described in [150]. Once a packet scheduled for translation arrives at the middlebox, a random number is created that serves as the randomization part. The random number XORed with the network prefix is hashed and the most significant 80bits are used as the new randomized subnet and interface identifier. This process is different to the one proposed in [9], which only randomizes the least significant bits of the host. Finally, 16bits (the length of the IP header checksum [14]) of the so-created IPv6 address have to be adjusted to make sure the checksum of the privacy-conform IPv6 addresses matches the checksum of the original packet. Luckily, valid Internet checksums can be calculated incrementally [126], thus, according to [159], PrivMid6 calculates the difference of the old and the new checksum and adds the value to a randomized position of the least significant 80bits.

Translation and Validity of Addresses

With the presented algorithm above, the middlebox is able to generate new randomized IPv6 addresses in a checksum-neutral way. The remaining question is which state has to be maintained at the middlebox and what happens if the lifetime of an entry expires.

With state of the art privacy extensions the default lifetime of an IPv6 address is 24 hours and can be adjusted as the only parameter by the user. However, smaller timeouts may result in dropped connections, which is one of the reasons that privacy extensions "should be disabled by default in order to minimize potential disruptions" [109].

The default behavior of PrivMid6 generates a randomized IPv6 address to every newly seen internal address and assigns a default lifetime of 24 hours to it (just like privacy extensions). As long as the internal address doesn't change, the external address remains identical for 24 hours and changes afterwards by generating a new address using a new random value. This behavior also drops existing connections since the middlebox is not aware of the actual state of upper layer protocols. Therefore, PrivMid6 also supports the tracking of TCP connections just like NAT44 or netfilter, allowing the assignment of individual lifetimes for IP 5-tuples. An example policy for a PrivMid6 enabled middlebox that maintains a global prefix of 2001:a200:f000::/48 may be as follows:

fixed prefix length	48
fixed subnet	1234
TCP destination port 80	per connection
default lifetime	2 hours

This policy specifies that IP addresses carrying the subnet 1234 are not translated, but only forwarded. These IP addresses can be used by hosts in the internal network to provide services and to run applications that require static IP addresses, such as VoIP and other peer-to-peer applications. All outgoing connections to TCP port 80 should be tracked and a new randomized IP address should be assigned to every new http request. For all other connections a lifetime of 2 hours is used. The fixed prefix length of 48 specifies that only these bits remain unchanged and for all other bits (subnet and interface identifier) randomization is desired.

8.3.5 Evaluation and Discussion

PrivMid6 aims at protecting hosts from being tracked based on their globally unique IPv6 address. Compared to the state of the art, PrivMid6 not only randomizes the interface identifier, but also additional (subnet) bits of the IPv6 address. We also target the problem of hosts moving through different networks that carry distinct prefixes, a scenario that is not solved by the IPv6 privacy extensions. Here we follow the approach of [9] by using the prefix as an input for the hash function.

Our threat model covers potential attackers aiming to correlate IPv6 addresses with clients by monitoring connections coming from the same network, e.g. by logging addresses on web-servers located in the Internet. For such attackers it should not be possible to establish a correlation between two different addresses by the means of being able to tell if two connections carrying two different IPv6 addresses were made from the same client. We assume that a potential attacker does not have access to the internal network, which would allow tracking hosts based on internal IP addresses or MAC addresses.

The security of our approach is dependent on a secure random generator and a secure collision resistant hash function. The adjustment of the checksum does not have any impact on the security of our system since only the already randomized address is changed. If an attacker is able to restore the original address before the checksum

adjustment, a correlation between the random and the original IPv6 address of the client is not possible.

PrivMid6 does not consider tracking on higher layers (above layer 3). Cookies [88], browser characteristics[10] and JavaScripts embedded in websites allow the tracking without considering the IPv6 address. The authors of [62] show how to track users based on DNS. We also do not consider the tracking of internal addresses by attacks that have access to the LAN segment. This also complies with our requirement R4, which aims at providing a possibility for legal interception. Also, the tracking based on the allocated prefix is still possible. However, all clients sharing one Internet access hold the same prefix, which is similar to today's approach of sharing one global IPv4 address among a number of clients using NAT44.

	IPv6	**PrivacyExt.**	**NAT66**	**PrivMid6**
Anonymizing	−	+	−	+
Topology Hiding	−	-	+	+
Protocol Independency	+	+	+	+
Legal Interception	+	-	+	+
Incoming Connections	+	+	+	+

Tab. 8.2: Comparison to the state of the art

Table 8.2 shows the advantages of our middlebox approach compared to the state of the art. For each requirement as defined above the middlebox provides a solution. R1 asks for anonymization, which can be provided via IPv6 privacy extensions or by our middlebox service. Here, PrivMid6 not only randomizes the interface identifier, but also additional bits of the subnet. PrivMid6 also provides topology hiding (R2) similar to NAT44: all internal clients carry the same prefix, while the suffix cannot be used to distinguish between different clients. This is in particular true when configuring a stateful translation with a per-connection obfuscation. Protocol independency (R3) is achieved by translating packets in a checksum-neutral way. Therefore, protocols above layer 3 remain unchanged. R4 (legal interception) is reached by the (optional) possibility of logging mappings between (fixed) internal addresses and their random external ones. Thus, a privileged user is able to suspend privacy, which of course may also be done by a potential attacker. By the definition of fixed addresses that are only forwarded and not translated, PrivMid6 achieves R5 and allows incoming connections.

Finally, we compare the performance of PrivMid6 to the performance of legacy NAT44 devices. Middleboxes that process packets introduce an additional delay to the network that is dependent on the number and the complexity of the individual processing steps. We distinguish between two scenarios: one, the processing of new packets and two, the processing of packets belonging to an already existing state. For packets that can be translated according to their state there is not much difference between NAT44 and PrivMid6. The mapping table is looked up, certain header fields are translated and the packets leave the outbound interface. However, PrivMid6 only needs to translate the IPv6 source address of outgoing packets, whereas NAT44 may also translate the source port (if not using port preservation). Additionally, PrivMid6

[10] https://panopticlick.eff.org/

only operates on layer 3 and translates in a checksum-neutral way, while NAT44 always has to adjust the checksum of the IPv4 and layer 4 header. Adjusting the checksum can be done incrementally [126] and only has little influence to the performance. For new packets both middleboxes need to allocate a new state before translating packets. NAT44 chooses a new external endpoint, while PrivMid6 needs to generate a randomized IPv6 address by applying a hash function to a random number. Once this new endpoint/address is generated, the middlebox continues with the address translation (step 6 of the processing model as shown in table 8.1).

8.3.6 Summary

This section targeted the privacy vs. connectivity problem that is introduced by globally unique IPv6 addresses. On the one hand, unique addresses re-establish the end-to-end paradigm and avoid many problems that have been observed when introducing NAT44. On the other hand, IPv6 addresses enable the tracking and identification of users across multiple (web)-sites and also allow creating movement profiles of mobile users. State of the art solutions, such as the IPv6 privacy extensions, operate on the hosts and randomize the last 64bit of an IPv6 address. Our privacy preserving middlebox service PrivMid6 allows specifying policies for packet flows, defining a trade-off between reachability and privacy. Privacy and topology hiding for IPv6 is established by randomizing IPv6 addresses and translating them in a checksum-neutral way. Here, not only the device identifier, but also additional subnet bits are randomized to provide a similar protection level as NAT44. Services can still be operated by defining certain subnets to be directly reachable and not to be handled by the translation function.

8.4 Virtual and Dynamic Infrastructures

New applications, services and communication paradigms have new demands regarding the underlying network infrastructure that often cannot be satisfied with existing deployments. Innovations, such as switching to a new network layer protocol, are hard to introduce to existing networks and ISPs avoid modifications to their infrastructure to protect their services and to limit costly investments. Middleboxes are one common way of providing new services to networks without having to change too many entities. There is one more paradigm that is currently discussed in the research community that hasn't been mentioned yet: a clean slate approach to the whole Internet infrastructure as described in [127] with the goal of establishing a new architecture without considering the currently deployed infrastructure. The migrating to a clean slate architecture, however, is costly and might only happen within restricted environments. Projects such as GENI[11] run virtualized instances of new architectures on the currently deployed network. Virtualization in general is a powerful tool for adding more flexibility and security to existing networks and for realizing new communication paradigms.

This section presents an architecture that combines host virtualization (via Virtual Machines (VM)) and network virtualization (via Virtual Private Networks (VPN)) to deploy dynamic infrastructures on top of existing networks. Virtual machines can be bridged to virtual networks, while the resource management can either be done by providing configuration files or by triggering decentralized configuration mechanisms, e.g. by hardware tokens like smartcards. We show how this architecture extends our MicroCA approach by implementing a middlebox service for clients located in unmanaged networks.

First, section 8.4.1 starts by explaining targeted application areas. Section 8.4.2 then presents our approach for combining host and network virtualization and describes the components of our architecture. Section 8.4.3 introduces VPN networks with a special focus on P2P-based solutions. We then present our P2PVPN solution that is based on Datagram-TLS (DTLS) [124] for authentication and uses the middlebox traversal framework NOMADS. Finally, section 8.4.4 shows the integration and configuration of our architecture.

8.4.1 Application Areas

Our proposed architecture mainly targets unmanaged networks, but can also be applied to enterprise networks. For unmanaged networks, such as home networks, our approach allows securely creating, deploying and running different services that are separated by virtualization. Separation is not only done by the means of hosts, but also by allowing to configure multiple virtual networks. Possible use-cases not only include self-configured services that benefit from separation, but also commercial services that require a restricted environment. A pre-configured home router may serve as a common service platform that enables different stakeholders to run their services. For example, a bank would distribute smartcards to their customers that are used (once inserted into the home router) to automatically download a virtual image, connect it to the bank's VPN network (which may be a traditional client-server based VPN) and execute banking services only within the protected environment. Services and network configurations on the home router can be extended to physical hosts by providing a middlebox-like service, e.g. by tunneling certain traffic to certain P2P networks.

[11] http://www.geni.net/

For enterprise networks our approach allows the deployment and management of flexible large scale networks. Unused resources on machines may be used for running virtualized services and dynamic networks can be configured and deployed from a central configuration file. Scenarios such as building networks call for multi-tenancy with many shared (computing) resources. Our approach can be seen as a foundation to dynamically configure (e.g. time or location dependent) IT services for shared building resources such as meeting rooms and offices.

8.4.2 Approach

We propose an architecture that is based on a combination of host and network virtualization. Different virtual machines can be in different virtual networks, thus providing isolation in multiple dimensions. Network virtualization (via virtual private networks) is only done within the host operating system (privileged domain), which provides great flexibility for managing large installations since reconfiguration can completely be done within one domain. The actual virtualized guest system only sees its network adapter and does not have to care about virtualization issues. Since the bridging is done in the privileged domain, the virtualized guest system is still able to configure its interfaces as without VPN. This is similar to the functionality that is provided by Virtual LAN (VLAN) in Ethernet.

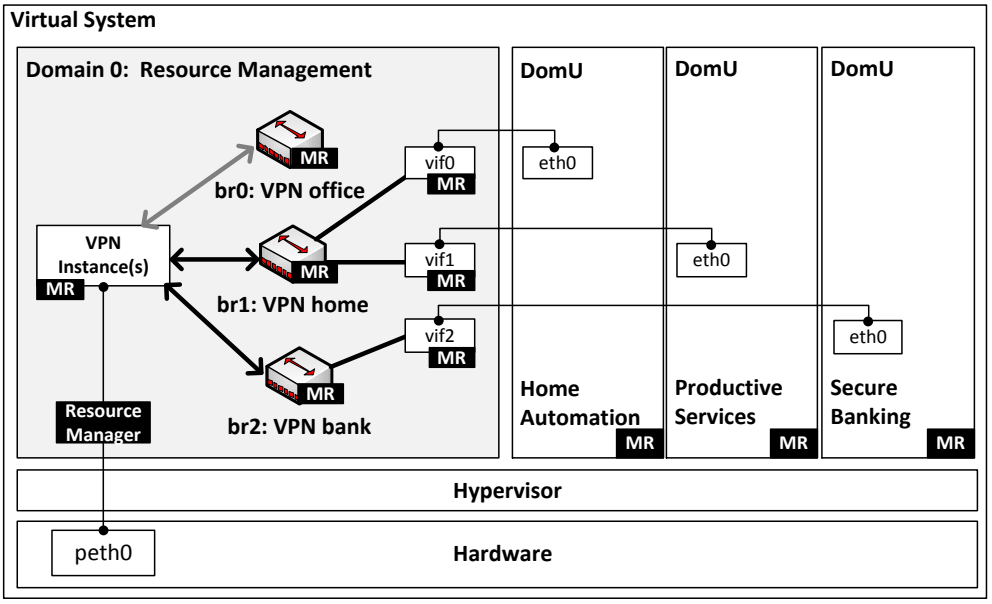

Fig. 8.7: Architecture for combining host and network virtualization

Figure 8.7 shows a reference example of our architecture. For the sake of security[12] the three user domains (DomU) for home automation, productive services and secure banking are isolated from each other. To directly and securely connect the according virtual machine to the banking server, a VPN instance in the Dom0 is bridged to the virtual representation of the banking machines network interface (vif2). The other two domains (home automation and productive services) are bridged to the home's VPN

[12] Isolation as provided by the utilized virtualization techniques has to be secure and virtualized guest systems should not be able to break out of their environment. For this chapter we assume that such techniques will exist in the future and will not target the security of today's hypervisors.

network, whereas the office network is currently not bridged to any virtual machine.

To configure the network interfaces, VPN instances and bridges, each resource is managed via a Resource Manager (RM). Each resource has its dedicated Managed Resource (MR) interface that is used by the RM to pass resource specific configurations to it. The RM itself is an accessible network service that acts as an interface to the complete virtualized system (host and guest machines) and is reachable via a network interface. Thus, the complete virtualized architecture can be seen as a manageable resource and can be dynamically set up and configured from a centralized management host located in another network.

In the following sections we describe the individual components in more detail. First, our solution for a P2P-VPN networking instance is presented. It is based on a structured P2P network and explicitly supports multiple virtual networks for one physical machine. We then present our resource manager and show how the components interact.

8.4.3 Virtualized Networks - P2PVPN

Virtual Private Networks (VPN) are overlay networks establishing a network topology over the physical one. VPNs are often used to interconnect different branches of one company or to securely join a network from remote. This allows to actually become part of the network, which includes a valid IP address for that network and the ability to broadcast into the network.

In a typical installation a centralized server manages authentication and authorization issues and acts as a single connection point. All traffic between the nodes travels through the server and is re-encapsulated and re-encrypted according to the client-based session keys. The bandwidth is shared among the clients and the server can be seen as a single point of failure for the whole system. A slightly different approach is based on peer-to-peer networks: Instead of letting a server distribute data packets among its clients, the peers directly connect to each other by forming a meshed network. For unmanaged networks this has an enormous advantage as the performance of a connection does not depend on the bandwidth of a single server, but only on the own and the remote Internet connection. Especially if we assume an asymmetric access to the Internet (meaning download vs. upload bandwidth), a centralized VPN access server located in a home network suffers from the performance.

However, there are many problems that have to be solved for P2P-VPN networks, such as authentication, authorization, revocation of access rights, confidentiality, middlebox traversal and bootstrapping just to name a few of them. This section presents different approaches to P2P-VPN, explains advantages and disadvantages and finally presents our own solution of a structured P2P-VPN System called P2PVPN.

Related Work

Many proposals have been made for establishing VPNs in a centralized and decentralized manner. We not only differentiate between the layers they work on (network (IPsec) vs. transport (OpenVPN)), but also how they provide certain functionality like traffic flow, configuration and authentication.

Table 8.3 follows the definitions given in [164] and shows five different approaches for providing VPN functionality. *Centralized VPN Systems* such as OpenVPN[13] rely on a (single) server that authenticates clients and routes traffic among all hosts in the

[13] http://www.openvpn.org

	Centralized VPN Systems	Centralized P2P Systems	Decentralized P2P Systems	Unstructured P2P Systems	Structured P2P Systems
Traffic	C/S	P2P	P2P	P2P	P2P
Routing	Server	Server	static	unstructured	DHT
Configuration	Server	Server	manual	connect to supernode	connect to DHT
Authentication	Server	Server	Peers	Peers	Peers
Example	OpenVPN	Hamachi Wippien	Tinc	N2N	GroupVPN **P2PVPN**

Tab. 8.3: Different approaches for VPN [164]

system, which means the bandwidth of the server is shared among all clients. This architecture is useful for typical client-server installations where a company provides access to its network via a VPN server located in their datacenter. With *Centralized P2P-VPN Systems*, a centralized server takes care of the authentication, while the actual data transfer is done peer-to-peer. This is a similar approach to Skype[14] and early P2P systems such as Napster[15] (discontinued due to legal issues and relaunched as a new service). The advantage is the management of centralized identities, which makes revocation and accounting easy. Centralized P2P systems still represent a single point of failure. One of the first P2P-VPN systems was the closed source solution Hamachi[16]. Later on, open source systems following the same approach, such as Wippien[17], appeared and are predominantly used for online games that require layer 2 broadcasts. In *Decentralized P2P-VPN Systems* there is no distinction between the peers, each one is able to provide the same functionality and is able to connect to others. Discovery and configuration of links is not defined and has to be done manually. *Unstructured P2P-VPN Systems*, such as N2N[18], provide peer-to-peer discovery, communication and authentication. Discovery is usually done using flooding or broadcasting in an unstructured manner. In N2N, each peer connects to a super-peer that is responsible for storing and broadcasting messages. However, N2N has a number of critical security flaws since it relies on a single shared key and does not provide authentication. *Structured P2P-VPN Systems* use a discovery mechanism that performs significantly better than flooding, usually by using Distributed Hash Tables (DHTs) and a lookup performance of $O(log\ n)$ (n being the number of nodes) or better for routing. GroupVPN [165] is an example for a structured P2P-VPN that uses DTLS [124] for authentication

[14] http://www.skype.com
[15] http://www.napster.com
[16] https://secure.logmein.com/products/hamachi/
[17] http://www.wippien.com/
[18] http://www.ntop.org/products/n2n/

and encryption. It is based on the IPOP (IP over P2P) framework [55] and allows easily creating and managing groups. Our work is inspired by GroupVPN and also uses DTLS for securing data communications. It uses the middlebox traversal framework developed in this thesis and an arbitrary storing mechanism only providing *GET* and *PUT* operations. Finally, allowing multiplexing multiple networks is an explicit requirement.

Key	**Value**
virtual MAC address + Group Name	public Endpoint
virtual IP address + Group Name	virtual MAC address

Tab. 8.4: Key/Value pairs for the P2PVPN network

Approach

The VPN instance has to provide authentication, encryption, encapsulation and forwarding of data packets. P2PVPN works as a bridged network handling layer 2 packets. Thus, we also need to take care of resolving layer 3 addresses to layer 2 addresses, which is done by the ARP protocol [117]. Instead of broadcasting ARP request like N2N, we propose to store the mapping in a (distributed) database and query it on an ARP request.

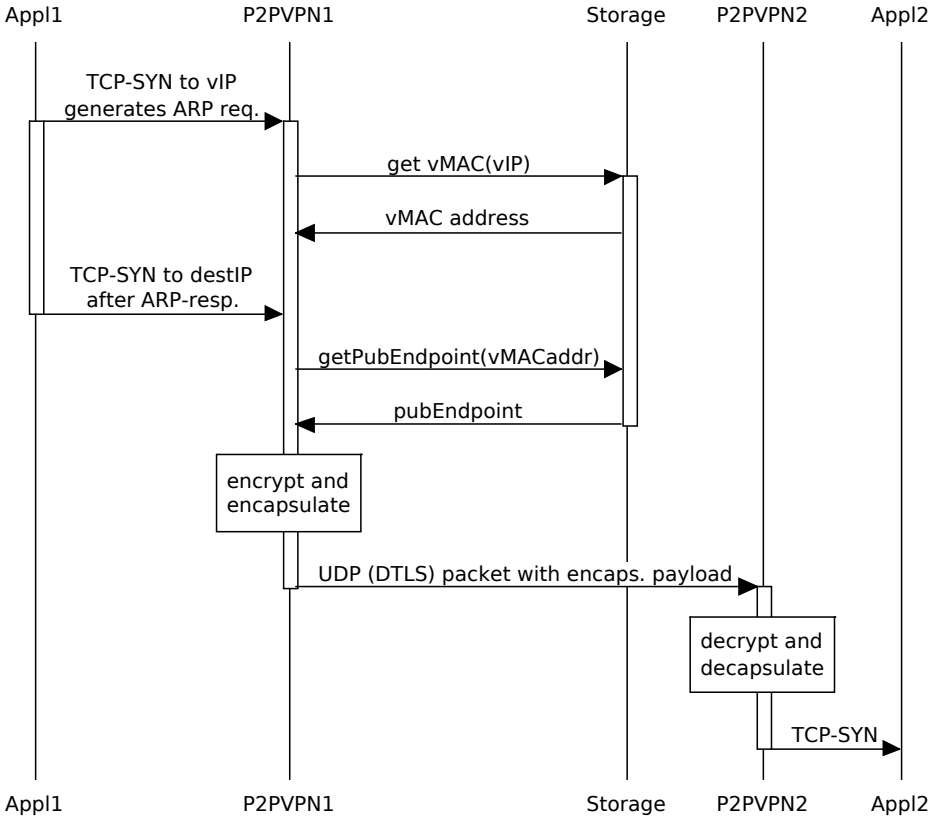

Fig. 8.8: Sequence diagram for the P2PVPN system

More specifically, we propose to use an arbitrary storing mechanism (SM) that holds $key \rightarrow value$ pairs and provides *PUT* and *GET* operations for manipulating them. Table 8.4 shows the key-value pairs that P2PVPN needs to store. Figure 8.8 then depicts the processing of outgoing packets. Once an application sends a TCP-SYN packet to a virtual IP address the operating system will generate an ARP request asking for the corresponding MAC address. P2PVPN will intercept this request and use the virtual IP address concatenated with the group name (and hashed again) as a key to the SM. The SM then returns the virtual MAC address of this host and P2PVPN is able to generate an ARP response. The initial TCP-SYN is passed to the P2PVPN process, which will lookup the actual endpoint of the remote peer. Therefore, the virtual MAC address together with the group name is used as a key in order to retrieve the value of the public endpoint. Finally, P2PVPN establishes a DTLS connection to the remote endpoint, which provides authentication and encryption of the encapsulated data packet. On the receiver side the packet is decrypted, decapsulated, written to the corresponding network device and eventually received by the application.

Security

Our security concept relies on X.509 certificates for authentication and uses DTLS as a protocol providing mutual authentication and data confidentiality over UDP. With the extension as defined in 7.6.1, DTLS in combination with our MicroCA is also able to provide authorization and allows restricting a connection based on additional metadata of the client. The distribution of certificates to clients, as well as the trust establishment between different networks can be done using all our mechanisms as described above.

Since P2PVPN can be seen as a purely decentralized approach, the security of the system depends on the security of the storing mechanism as described above. For unmanaged networks and small groups as created by our MicroCA approach, the storing mechanism can be kept private and very simple (e.g. a protected web-service or database running on a home router) and more or less static configurations can be exchanged via a secure channel (secured by the MicroCAs) before establishing the actual P2PVPN connection. This avoids security problems of storing mechanisms such as Distributed Hash Tables (DHTs). However, for larger deployments DHTs might be a valid design choice.

Additionally, there are a few extensions when considering the fact that there is no central authority in the network. Before joining a centralized VPN network, a new client has to authenticate itself to the VPN server. In P2PVPN we follow two different approaches: the first one assumes that one peer acts as a central authority and decides who is allowed to join the network (monarchy approach [46]). This super-node would therefore run the PDP and define policies for it. Additionally, the initiator might sign all entries before adding them to the storing mechanism.

Our second approach follows the group voting approach. Whenever a new client wants to join the network, members (either all or only a subset) are allowed to vote if they want to accept the new node or not. As one important decision factor, in [41] we utilized smartcards following the Java Card 3.0 Connected Edition[19] standard to check the integrity of a system (by performing a remote attestation [152]) before letting it join the VPN. This allows to make sure that clients within the virtual network are not infected by malware or viruses and only run well-defined and well-understood software.

[19] http://www.oracle.com

Implementation

Our implementation is based on a TAP device that forwards all traffic including the layer 2 headers to the user-space of our program. We use the patched openssl library[20] for DTLS operations and the DHT Entangled[21] based on Kademlia [94] as an example for a storage mechanism.

There are basically two possibilities for multiplexing incoming and outgoing VPN connections: First, it would be possible to define a small header that carries the group name, which allows using only one transport layer port for all virtual networks. Second, and this is the approach we follow, multiplexing can be done using different ports to distinguish different virtual networks. Thus, once a new P2P instance is set up, an available port is picked and registered to the DHT.

The algorithm below (8.2) shows the pseudo-code for the processing of outgoing packets following the steps of figure 8.8. For the integration of the middlebox traversal framework we provide two possibilities: First, a peer may choose to register the P2PVPN endpoint as a globally reachable service (middlebox service category Global Service Provisioning (GSP)) and thus call the framework to make the endpoint publicly reachable. Second, if signaling is desired the peer may register an identifier (e.g. its unique cryptographic ID according to 7.4.1) that can be used to agree on a valid public endpoint following the protocol as presented above.

```
1  while receiveFromTAP do
2  |  analyzeLayer2(packet);
3  |  if packet is ARP then
4  |  |  intercept ARP;
5  |  |  query cache or DHT for vMAC;
6  |  |  assemble ARP response and send it;
7  |  else
8  |  |  analyzeLayer3(packet);
9  |  |  query cache or DHT for public endpoint;
10 |  |  if remote endpoint then
11 |  |  |  invoke middlebox traversal;
12 |  |  end if
13 |  |  pass packet to DTLS ;
14 |  end if
15 end while
```

Alg. 8.2: Pseudo-code for the processing of outgoing packets

Evaluation

Finally, we evaluate the performance of our implementation as it is critical for the applicability of our approach and compare it to the state of the art VPN solutions Hamachi, N2N and OpenVPN. Most VPNs are implemented in the user-space of the operating system and context switches between kernel and user-space decrease the maximum throughput. Additionally, encryption and encapsulation has to be performed for every packet, which also influences the processing time.

The experiments were conducted on three PCs with a standard Debian GNU/Linux 6.0 operating system running kernel 2.6.32-5-amd64. Two clients with a quad-core Intel

[20] http://www.openssl.org
[21] http://entangled.sourceforge.net/

Core i5-2520M CPU and 8Gb of RAM were connected to a gateway with two Intel Gigabit network cards and an AMD Athlon(tm) Dual Core Processor 4850e with also 8Gb of RAM. The gateway acted as a server when evaluating OpenVPN, as a supernode for evaluating N2N and as the DHT for P2PVPN. There was no cross-traffic and the roundtrip time of ICMP echo requests/responses between the two clients was $0.406ms$ on average. The performance measurements were done using the Linux tool iperf[22] with its standard settings. Each test was repeated 50 times. The direct average throughput (10 runs with a standard deviation of 1.6) between the two clients was $933MBit/s$.

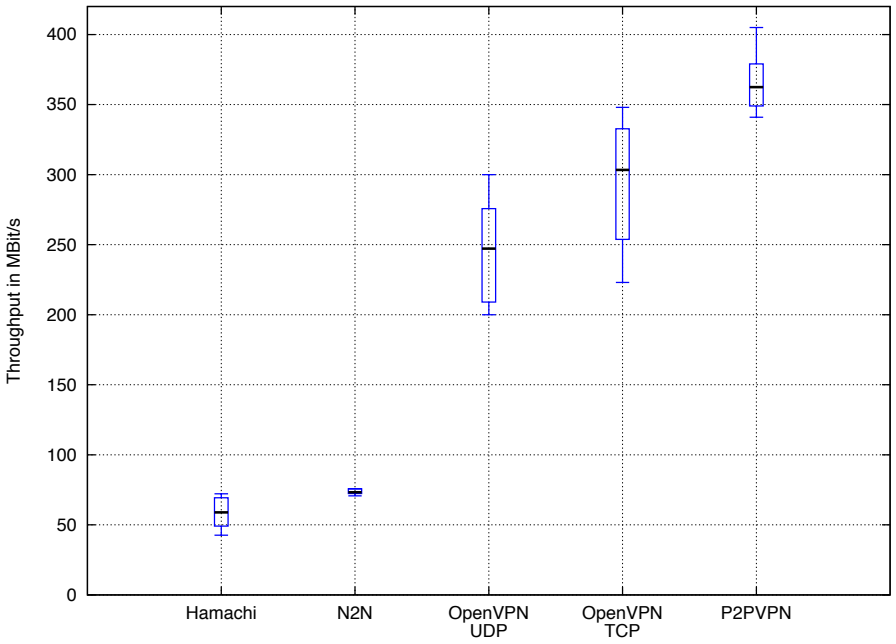

Fig. 8.9: Performance of P2PVPN compared to related (P2P)VPN implementations

The first test compares P2PVPN with a selection of the state of the art, namely Hamachi, N2N and OpenVPN. The results are depicted in figure 8.9, where each bar depicts the maximum and the minimum value of all measurements. The rectangle shows the range between the 0.1 and the 0.9-quantile (80% of all values). The black line within the rectangle represents the arithmetic mean value for each test series. For Hamachi the standard settings were used, which led to an encapsulated UDP traffic between the two clients. With an arithmetic mean value of $58.91MBit/s$, Hamachi was the slowest of our candidates. As the Linux version is only in a beta state and the source code is not available, we were not able to analyze this further.

N2N[23] was configured with a static key passed at the command line and used a supernode running at the gateway. The supernode was only contacted by the edge nodes at the beginning of the connection to determine the actual endpoints and for resolving ARP requests via broadcast messages. Afterwards, the packet flow was from client to client via UDP. With these settings we measured an average throughput of $73.28MBit/s$ on our gigabit link.

We then set up an OpenVPN server on the gateway in order to compare P2P-VPN approaches to traditional client-server implementations. We distributed certificates to our clients and configured them to use the AES-128-CBC algorithm for encryption and

[22] http://sourceforge.net/projects/iperf
[23] svn trunk, checkout August 5th 2012

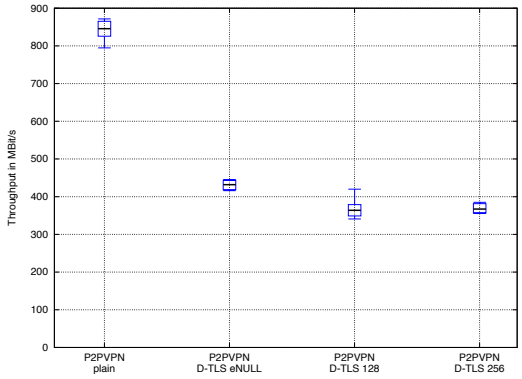

Fig. 8.10: Performance of P2PVPN

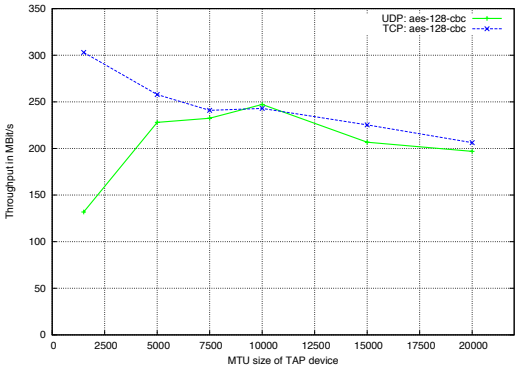

Fig. 8.11: Performance of OpenVPN depending on the MTU size

enabled LZO compression (disabling compression didn't influence our results). The MTU of the TAP device was set to 1500. We allowed a peer-to-peer communication by setting the client-to-client option. With these settings we got a throughput of only $131MBit/s$ for UDP encapsulation and an average of $303.4MBit/s$ for TCP encapsulation. When adjusting the MTU size, we finally got a maximum throughput of $247.2MBit/s$ for UDP. Figure 8.11 shows the influence of the MTU value on the throughput. By setting the value to 10000 and by disabling OpenVPN's internal fragmentation, the throughput increases dramatically on a gigabit link. A $100Mbit/s$ Ethernet link can be saturated by the default values. This is also documented on the OpenVPN website[24].

Finally, P2PVPN was configured to use DTLS with X.509 certificates and AES-128-CBC as its encryption algorithm. Our implementation uses the standard Debian OpenSSL library version 0.9.8o-4 that supports DTLS. The DHT was configured on the gateway and the clients used it only to resolve ARP messages and to look up the remote address once. With these settings we measured an average throughput of $362.5MBit/s$. The advantage of sending traffic peer-to-peer is because the OpenVPN gateway decrypts and re-encrypts every message that is forwarded from one client to another. With P2PVPN, a DTLS connection is directly established between two peers. We confirmed this finding by measuring the throughput of OpenVPN between one client and the server: with $350.4MBit/s$ it was almost as fast as P2PVPN between two clients.

We then evaluated P2PVPN in more detail by adjusting the encryption parameters of DTLS. Figure 8.10 shows the results of four measurement series. *P2PVPN plain* bypassed the DTLS library and our rather rudimental thread handling. Iperf packets were simply encapsulated, sent via UDP and decapsulated again, leading to a throughput of $845.75MBit/s$. When enabling DTLS again and setting the encryption method to $eNULL$ (no encryption at all) the throughput dropped to $431.92MBit/s$. Thus, threading and the overhead of the DTLS protocol has a significant impact on the performance. We then compared two different encryption settings: with a key length of 128bit we obtained a throughput of $362.5MBit/s$ as described above. When setting it to 256bit, the throughput remained at the same level: $367.11MBit/s$, which meets our expectation for a modern quad-core CPU. Considering the early stage of the DTLS implementation and our proof-of-concept implementation, there is further room for improvement in the future. Our throughput measurements show that a peer-to-peer

[24] http://community.openvpn.net/openvpn/wiki/Gigabit_Networks_Linux

VPN solution is suitable for asymmetric Internet services that can often be found in unmanaged networks. The latency of establishing new connections, however, depends on the storage mechanism or signaling infrastructure (e.g. the DHT) and will most likely be higher for P2P approaches. The actual data connection benefits from direct connections between individual peers, which removes the need for the re-encapsulation of packets on the server.

8.4.4 Resource Manager

After choosing one of the VPN solutions as described above for virtualizing networks, the Resource Manager (RM) is contacted to deploy the new configuration or to manipulate an existing one. We propose to use our unique cryptographic identities as described in chapter 7 for identifying the RM and for connecting to it. The RM can then be seen as a legacy SSL/TLS service and equipped with the ability to also query a PEP/PDP for authorization. In a nutshell, our resource manager has the following properties:

- Each RM holds a valid certificate signed by the corresponding MicroCA.
- The ID of the RM (*ServiceID.MicroCAID*) together with its current IP address is registered to a database for reachability.
- The RM uses our middlebox traversal framework and registers its current IP and port as a globally reachable service for reachability.
- The RM uses SSL/TLS to 1) mutually authenticate clients and 2) intercept the handshake for authorization.
- (XACML)Policies in the MicroCA are used to configure who is allowed to access what functionality. For example, the delegation feature of XACML version 3.0 allows an administrator to delegate the configuration of a resource (e.g. a VPN for gaming purposes) to a specific user.
- The interface of the RM towards the administrator is rather generic and follows our approaches for bridging heterogeneity in home networks as presented in [112]. The RM itself is responsible for managing host specific resources, e.g. invoking the bridging command in Linux vs. bridging commands in other operating systems.

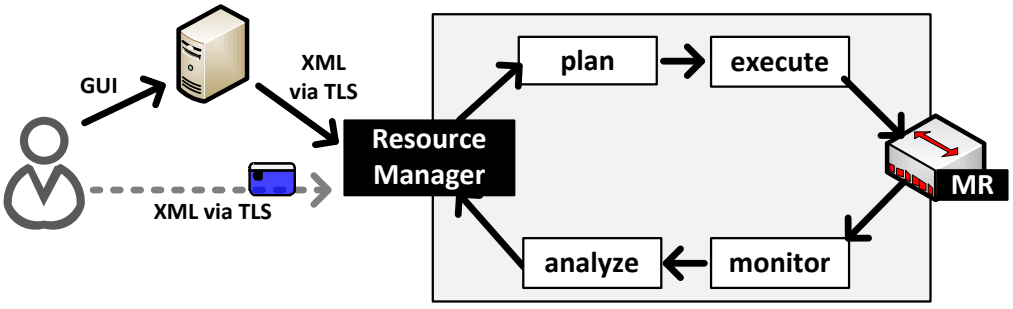

Fig. 8.12: Concept of the Resource Manager implementing MAPE

The internal concept of the RM is derived from approaches to autonomic computing [54]. The RM implements a closed control loop (MAPE: Monitor, Analyze, Plan, Execute) enabling situation-based decisions and feedback [39] to control Managed Resources (MR) such as network interfaces, virtual machines and bridges. As depicted

in figure 8.12, a user interacts with the RM either via a GUI or by directly sending a configuration file to it. The rather generic description (see listing below) is then transformed within the *plan* stage into an implementation specific command. For example, an "add VM *office* to P2PVPN network *office*" directive would be transformed to the Linux command "brctl addif office vif0", thus adding the virtual interface *vif0* to the (already existing) bridge named *office*. The command is then invoked within the *execute* stage, while the monitor stage continuously observes changes and reports them back in a structured way (*analyze*) using our software tool LinEX [106] as described in section 4.3.5.

```
<machine_definition>
  <libvirt>
     <description>
        <domain type="xen">
         <name>machine1</name>
         <uuid>f0672baf-3dbd-6923-3e68-d1a3d1af568a</uuid>
         <memory>131072</memory>
         ...
         <devices>
            <disk type="file" device="disk">...</disk>
              ...
         </devices>
        </domain>
     </description>
  </libvirt>
  <p2pvpn>
      <param name="groupName">home</param>
      <param name="IP">172.16.1.1</param>
      <param name="MAC">00:34:11:a5:32:ba</param>
      ...
  </p2pvpn>
</machine_definition>
```

List. 8.2: Example configuration file for the RM

Listing 8.2 shows an example configuration file of a virtual configuration as a user would send it (via the SSL/TLS service) to the RM. In order to describe the settings of the virtual machine itself, we decided to use libvirt[25], a virtualization API that is able to utilze many different virtualization techniques such as XEN[26], KVM[27], VMware[28] and OpenVZ[29]. In addition to the VM description we defined a P2PVPN section that describes to which virtual network the VM should be bridged. In this case, the VM *machine1* is part of the virtual network *home*. Thus, the *analyze* stage of the RM has to determine if the network already exists and decide which commands should be invoked. For example, if the virtual network already exists, it would simply add the virtual interface of *machine1* to the bridge. If it doesn't exist, a new P2PVPN instance, as well as a new bridge has to be created. The same is true for leaving the network or for joining a different one. This example illustrates the usefulness of the separation of the *analyze* and *execute* stage.

[25] http://libvirt.org
[26] http://www.xen.org
[27] http://www.linux-kvm.org
[28] http://www.vmware.com/
[29] http://www.openvz.org

On using Smartcards for Configuration

Instead of relying on an external entity that provides the configuration file, a smartcard can also be used to initialize the system. It may unlock (or even hold) policies for new configurations and once connected to the configuring host (e.g. an external management host or b directly controlling the Resource Manager of a machine) and unlocked by the user (a PIN is used for encrypting the private key that is stored on the card), an encrypted configuration file can be downloaded from a remote server and loaded into the RM. In a more advanced scenario the configuration file can also be stored on the smartcard itself, thus being completely independent from external entities. We developed and used this approach in [79] to automatically load and en/decrypt virtual machine images for distributed installations.

8.4.5 Summary

This section presented an approach that combines host and network virtualization allowing the deployment of dynamic infrastructures. A Resource Manager implementing a MAPE cycle and running on the host operating system (the privileged domain) configures the system based on a high-level configuration as provided by a user or by a hardware token. We also presented P2PVPN, a solution for bridging networks in a decentralized way using an arbitrary storing mechanism. We showed how this architecture can be used for extending our MicroCA approach for unmanaged networks and mentioned different application areas and future scenarios.

8.5 Summary and Key Findings

In this section we showed how middlebox services can be applied to open problems in today's networks (Q4 according to section 1.1). The first middlebox service equips the Devices Profile for Web-Services (DPWS) with the authorization framework XACML, thus securing the access to specific services in the network. Policies on the middlebox also allow controlling the discovery of services in remote trusted networks using our secure identities and trust relationships as presented in chapter 7. Our second middlebox service, PrivMid6, targets the privacy problem that is introduced by the deployment of IPv6. Instead of relying on hosts to implement IPv6 privacy extensions, PrivMid6 provides privacy to a network by translating between fixed internal and randomized external addresses in a checksum-neutral way. In contrary to state of the art solutions, PrivMid6 also randomizes the subnet bits and allows defining blocks of static addresses that can be used to provide services. Finally, our third middlebox services proposes to combine host and network virtualization to create and manage dynamic network infrastructures for home and enterprise networks.

Key Findings of this chapter
(and contributions according to section 1.2):

- **Unmanaged networks offer only little security and privacy features. Middlebox services help to introduce new features to existing networks without having to change existing software deployments.**

C8.1 **We used the Devices Profile for Web Services (DPWS) as a reference example for connecting existing SSL/TLS-based services to our MicroCA approach. A new middlebox service allows discovering and using service in trusted remote networks and adds authorization to DPWS.**

C8.2 **Our middlebox PrivMid6 targets the privacy vs. connectivity problem of IPv6 networks by randomizing the complete IPv6 suffix based on policies. PrivMid6 provides a trade-off between privacy and reachability without relying on the hosts to run specific software.**

C8.3 **Virtualization helps to run new services on top of existing installations and networks. The proposed combination of host and network virtualization together with a P2PVPN approach allows deploying dynamic and flexible configurations for unmanaged networks.**

Part V

CONCLUSIONS

9. CONCLUSION

During the last few years the Internet has become the largest communication network in the world. The growing commercial success of the Internet, the availability of more and more devices and the need for innovations have led to the deployment of middleboxes and to the violation of the initial end-to-end argument. This violation causes numerous problems: Hosts located in private networks are not directly reachable from the Internet, packets are filtered by firewalls and proxies and web-caches only deliver specific web-content.

The goal of this thesis was to understand the behavior of middleboxes and to develop solutions to cope with them. In part I we gave an introduction to this thesis and presented the fundamentals regarding Internet architectures and middleboxes. Part II analyzed middlebox behavior by designing algorithms for an experimental analysis. The results of our conducted field test served as an input for part III where we developed our knowledge-based approach to middlebox traversal. Finally, part IV covered the security and application of middleboxes and presented a security architecture for unmanaged networks, as well as new middlebox services.

9.1 Contributions

The contribution of this thesis is the study and development of concepts and algorithms for understanding the behavior of middleboxes, their traversal, as well as their application to problems that have emerged with the growing success of the Internet. The main contributions of this thesis (as also shown in table 9.1) are as follows:

Processing and Information Model for Middlebox Behavior (C3.1, C4.1): Due to the lack of standardization, middlebox behavior influences many protocols and applications in a negative way. In the state of the art middlebox behavior has not been thoroughly analyzed in a way that helps to fully understand the implications for applications and traversal techniques. In this thesis we presented two models for describing middlebox behavior: Our processing model allows expressing the individual processing steps for arbitrary middleboxes. Our information model on the other hand holds behavioral characteristics and properties of middleboxes. An instance of the information model represents the exact behavior of a specific middlebox and can be used as an input for customizing traversal techniques. The key findings of chapters 3 and 4 regarding the processing and information model were as follows:

- Middlebox behavior and especially NAT behavior is not standardized.
- In the state of the art there have been many efforts to characterize NAT behavior, but the impact of these classifications remains unclear.
- Our processing model helps to describe and understand middlebox packet processing using arbitrary operations.
- Our information model contains relevant middlebox properties and serves as a formal representation for middlebox behavior.

Research Question	State of the Art	This thesis
Q1 Does a thorough and structured analysis of middlebox behavior help to understand the implications for applications and services and to improve the success rate of middlebox traversal techniques?	MB implication for applications unclear. Success rate of many techniques rather low.	**C3.1, C4.1-C4.3**: Experimental Analysis shows more room for improvement if behavior is understood.
Q2 Can the knowledge about the behavior of involved middleboxes be used to apply and parameterize middlebox techniques suitable for applications and the involved infrastructure?	Many individual solutions and frameworks with trial and error (ICE, Skype).	**C5.1, C6.1-C6.2**: Our knowledge-based framework parameterizes MB techniques based on MB behavior and on application specific requirements.
Q3 Can security mechanisms common in enterprise networks be applied to networks that lack professional administration in order to secure middlebox traversal?	Very little security mechanisms in unmanaged networks due to complexity.	**C7.1**: Our security infrastructure allows the management of secure identities in a user-friendly way and hides the complexity from its users.
Q4 Can well-designed and well-understood middlebox services be a valid design principle for the Internet and can middlebox services be applied to open problems in today's networks?	Many commercial middleboxes. Impact unclear.	**C8.1-C8.3**: Well-defined middlebox services for solving open problems proposed.

Tab. 9.1: Contributions of this thesis

Experimental Analysis of Middlebox Behavior (C4.2, C4.3): To understand different middlebox implementations we conducted an experimental analysis. Our designed test algorithms create instances of the information model that can be verified using a virtualized testbed. A public field test gave an insight on the deployment and behavior of middleboxes in today's Internet. Based on the results of the field test we were able to give recommendations on how to improve existing traversal solutions and how to design future ones. The key findings of chapter 4 were as follows:

- The test routines and algorithms designed in this thesis allow systematically measuring middlebox behavior and to create an information model instance that represents a specific middlebox. An experimental analysis first verified our algorithms in a virtualized testbed, before conducting a public field test.
- Only 68.4% (63.4% for TCP) of all constellations in the Internet provide the essential prerequisites for behavior-based traversal.
- Even if the middlebox implements a connection dependent binding strategy, in approx. 60% for UDP and 46% for TCP it is still possible to predict the external binding by analyzing binding patterns.
- Large Scale NAT is already deployed by many ISPs and hinders communication, especially in mobile networks.

- A single traversal mechanism can never be as effective as a solution that carefully parameterizes a basic traversal algorithm according to the current situation, the topology and the behavior of the involved middleboxes.

Service Categories for Middlebox Traversal (C5.1): When presenting the state of the art in middlebox traversal we figured that existing solutions not only have a low success rate, but also establish connections without considering additional requirements of applications and users. We stated that a middlebox traversal technique should enable an application to communicate across middleboxes in a way that is compliant with authorization and additional policies as defined by privileged users. This led us to the definition of four middlebox traversal service categories that also consider external infrastructure and the role of an application. We showed how existing traversal techniques can be applied to our service categories and stated that a middlebox traversal framework should select a technique based on the accessibility requirements of an application. The key findings of chapter 5 were as follows:

- Numerous state of the art traversal techniques exist that may or may not work dependent on the behavior of the middlebox and the availability of external infrastructures.
- A middlebox traversal technique should not only enable the communication across middleboxes, but also consider authorization and accessibility requirements, as well as additional policies as defined by a user or administrator of a network.
- Middlebox Traversal Service Categories help to achieve this goal by taking the role of the application, as well as available infrastructure into account.

Knowledge-based Framework for Middlebox Traversal (C6.1, C6.2): Based on the findings of our field test and on the evaluation of existing middlebox traversal techniques we presented NOMADS, an innovative framework for middlebox traversal. NOMADS not only applies middlebox traversal techniques for the sake of connectivity, but also considers higher-level requirements of applications. Decisions about the applicability of integrated traversal techniques are made based on the knowledge about the network, the application and the participating entities. Our information model helps to pair NOMADS instances and to find an optimal solution for the given situation. Besides being more flexible, the knowledge-based approach also has performance advantages over the state of the art due to decoupling the gathering of information from the actual traversal. Finally, two new middlebox traversal techniques were presented and the key findings of chapter 6 were as follows:

- Knowledge-based middlebox traversal was proposed to react to the findings of our field test.
- The NOMADS framework supports our middlebox traversal service categories and considers external user-defined policies.
- NOMADS integrates existing traversal techniques and parameterizes them based on the actual behavior to increase their success rate.
- NOMADS is significantly faster and more applicable to open problems than state of the art frameworks.
- Two new middlebox traversal techniques for restricted environments were developed: DPWS-IGD can be seen as a secure alternative to UPnP and AutoMID allows hole punching without the need of a third party.

Security Infrastructure for Unmanaged Networks (C7.1): Today, the authentication of services in the Internet is provided via digital public key certificates. Due to the complexity of maintaining a traditional PKI, authentication in unmanaged networks is often not available. We argued that an adaption to unmanaged networks is possible and identified the combination of a local Certification Authority, the MicroCA, and the Web of Trust approach as suitable for unmanaged networks. This combination enables the easy management of all devices, users and services within one network and to establish trust into a complete key-family rather than in only one key. Our system assists inexperienced users in maintaining their own PKI and hides the technical complexity from them. Trust establishment mechanisms for a direct pairing and a remote pairing via social networks were presented. The key findings of chapter 7 were as follows:

- Inexperienced users that are often found in unmanaged networks are not able and/or willing to maintain enterprise grade security mechanisms, although they would benefit from them.
- The presented MicroCA architecture provides almost zero-configuration plug and play security to unmanaged networks.
- The Device Registration process describes an easy-to-use mechanism for issuing public key certificates to devices and users.
- By using already established trust relationships of social networks in combination with a Web of Trust-like rating system, it is possible to establish a certain level of trust into a previously unknown identity.
- Our infrastructure provides authorization to legacy services through SSL/TLS interception.

Innovative Middlebox Services (C8.1-C8.3): Middleboxes may not only cause problems but, if designed and deployed in a well-defined way, may also be valid design components for today's and tomorrow's Internet. In chapter 8 we presented three middlebox services and our findings were as follows:

- Unmanaged networks offer only little security and privacy features. Middlebox services help to introduce new features to existing networks without having to change existing software deployments.
- We used the Devices Profile for Web Services (DPWS) as a reference example for connecting existing SSL/TLS-based services to our MicroCA approach. A new middlebox service allows discovering and using service in trusted remote networks and adds authorization to DPWS.
- Our middlebox PrivMid6 targets the privacy vs. connectivity problem of IPv6 networks by randomizing the complete IPv6 suffix based on policies. PrivMid6 provides a trade-off between privacy and reachability without relying on the hosts to run specific software.
- Virtualization helps to run new services on top of existing installations and networks. The proposed combination of host and network virtualization together with a P2PVPN approach allows deploying dynamic and flexible configurations for unmanaged networks.

9.2 Future Work

For each part of this thesis there are a number of directions for future research:

Behavior of Middleboxes: The first instance of the field test conducted in this thesis was dependent on volunteers that download a software and execute it on their computer. To remove the platform dependency, a web-based solution was developed. To reach a larger number of users further improvements are desirable: We have already implemented selected algorithms for an early version of the Android-based measurement infrastructure MeasrDroid, which allows constantly measuring the current network and delivers promising results. An adaption of all algorithms to Android is highly desirable. Additionally, new measurement platforms, such as the browser-based Fathom [32] or techniques such as web-sockets and Java Scripts, may help to collect more data. The individual tests and algorithms can be seen as a proof of concept and there is further room for improvement. Especially our topology detection algorithm may deliver more accurate results with a TCP-based implementation. Finally, we did not conduct further tests on content manipulation, filtering and censoring. These aspects could be examined by integrating appropriate test routines into our software.

Traversal of Middleboxes: Our knowledge-based approach was motivated by the findings of our field test and many of the findings have been adapted and integrated. However, there is still room for further improvements: The implication of multiple clients behind the same middlebox have only been considered marginally when presenting our endpoint prediction approach. However, for the coordination of clients located within the same network a more detailed knowledge about the topology is essential for a high success rate on middlebox traversal. Additionally, the use of a distributed signaling infrastructure (e.g. P2P-SIP) was proposed, but not implemented and evaluated. Future work should also look at different ways of utilizing existing infrastructures and to extend our AutoMID approach to be used for coordinating NOMADS instances.

Security and Application of Middleboxes: Our secure service infrastructure is an approach for introducing enterprise-grade security mechanisms to unmanaged networks. While we mainly focused on the technical side, future research should also consider the user's perspective, because the frustration of overstrained users is the main hindrance for the introduction of a security solution. Well-designed graphical interfaces and clear usability structures are essential for a real world deployment. Pre-defined policies that are distributed in an app-store like manner may also help to simplify setting up an initial installation. Furthermore, the extension of additional existing services is necessary to support a wide range of applications. One possible way is to introduce the proposed SSL/TLS interception to existing security libraries. Finally, the runtime integrity of the MicroCA and of our virtualized service platform should be ensured by host-based integrity measurements.

APPENDIX

A. DTD OF THE MIDDLEBOX INFORMATION MODEL

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.
    org/MiddleboxSchema" xmlns="http://example.org/MiddleboxSchema">

<xs:simpleType name="TableStrategyEnum">
  <xs:restriction base="xs:string">
  <xs:enumeration value="Replacement"/>
  <xs:enumeration value="Block"/>
  <xs:enumeration value="Other"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="TimerNameEnum">
  <xs:restriction base="xs:string">
  <xs:enumeration value="TCP-SYN"/>
  <xs:enumeration value="TCP-Established"/>
  <xs:enumeration value="TCP-FIN"/>
  <xs:enumeration value="UDP"/>
  <xs:enumeration value="UDPStream"/>
  <xs:enumeration value="Other"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="FilteringTypeEnum">
  <xs:restriction base="xs:string">
  <xs:enumeration value="Independent"/>
  <xs:enumeration value="Address Restricted"/>
  <xs:enumeration value="Adress and Port Restricted"/>
  <xs:enumeration value="Other"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="ProtocolLayerEnum">
  <xs:restriction base="xs:string">
  <xs:enumeration value="DataLinkLayer"/>
  <xs:enumeration value="NetworkLayer"/>
  <xs:enumeration value="TransportLayer"/>
  <xs:enumeration value="ApplicationLayer"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="NATBindingEnum">
  <xs:restriction base="xs:string">
  <xs:enumeration value="EndpointIndependent"/>
  <xs:enumeration value="ConnectionDependent"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="PoolingEnum">
  <xs:restriction base="xs:string">
  <xs:enumeration value="Paired"/>
```

```
51    <xs:enumeration value="Unpaired"/>
52    </xs:restriction>
53  </xs:simpleType>
54
55  <xs:simpleType name="PortBindingEnum">
56    <xs:restriction base="xs:string">
57    <xs:enumeration value="PortPreservation"/>
58    <xs:enumeration value="NoPortPres"/>
59    </xs:restriction>
60  </xs:simpleType>
61
62  <xs:simpleType name="MappingAlgorithmEnum">
63    <xs:restriction base="xs:string">
64    <xs:enumeration value="Fixed-Delta"/>
65    <xs:enumeration value="Pattern-based"/>
66    <xs:enumeration value="Error-Prone"/>
67    </xs:restriction>
68  </xs:simpleType>
69
70  <xs:simpleType name="FilteringProtocolBasedEnum">
71    <xs:restriction base="xs:string">
72    <xs:enumeration value="Echo Request out, TTL exceeded in"/>
73    <xs:enumeration value="UDP out, TTL exceeded in"/>
74    <xs:enumeration value="Echo Request out, Echo Reply in"/>
75    <xs:enumeration value="TTL exceeded out"/>
76    <xs:enumeration value="Echo Request out"/>
77    <xs:enumeration value="Echo Reply out"/>
78    <xs:enumeration value="Other"/>
79    <xs:enumeration value="TCP-SYN out, TCP-SYN in"/>
80    <xs:enumeration value="TCP-RST out"/>
81    </xs:restriction>
82  </xs:simpleType>
83
84  <xs:simpleType name="StatefulPolicyEnum">
85    <xs:restriction base="xs:string">
86    <xs:enumeration value="UDP out, Des. unreachable in"/>
87    <xs:enumeration value="UDP out, ICMP TTL exceeded in"/>
88    <xs:enumeration value="TCP-SYN out, TCP-RST in"/>
89    <xs:enumeration value="TCP-SYN out, ICMP TTL exceeded in"/>
90    <xs:enumeration value="Other"/>
91    </xs:restriction>
92  </xs:simpleType>
93
94  <xs:simpleType name="NoStatePolicyEnum">
95    <xs:restriction base="xs:string">
96    <xs:enumeration value="UDP in, Dest. unreachable out"/>
97    <xs:enumeration value="TCP-SYN in, TCP-RST out"/>
98    <xs:enumeration value="Other"/>
99    </xs:restriction>
100 </xs:simpleType>
101
102 <xs:complexType name="IPAddressTypev4">
103     <xs:sequence>
104         <xs:element name="IPAddress" type="xs:string"/>
105         <xs:element name="Netmask" type="xs:string"/>
106         <xs:element name="Comment" type="xs:string" minOccurs="0" maxOccurs="1"/>
107     </xs:sequence>
108     <xs:attribute name="IPVersion" type="xs:string" fixed="IPv4"/>
109 </xs:complexType>
110
111 <xs:complexType name="IPAddressTypev6">
```

```
112     <xs:sequence>
113         <xs:element name="IPAddress" type="xs:string"/>
114         <xs:element name="Prefix" type="xs:string"/>
115         <xs:element name="PrivacyExtension" type="xs:boolean" minOccurs="0" maxOccurs="1
                "/>
116         <xs:element name="Comment" type="xs:string" minOccurs="0" maxOccurs="1"/>
117     </xs:sequence>
118     <xs:attribute name="IPVersion" type="xs:string" fixed="IPv6"/>
119 </xs:complexType>
120 
121 <xs:complexType name="NetworkInterfaceType">
122     <xs:sequence>
123         <xs:element name="IPv4" type="IPAddressTypev4" minOccurs="0" maxOccurs="1"/>
124         <xs:element name="IPv6" type="IPAddressTypev6" minOccurs="0" maxOccurs="
                unbounded"/>
125     </xs:sequence>
126     <xs:attribute name="NetworkInterfaceName" type="xs:string"/>
127 </xs:complexType>
128 
129 <xs:complexType name="ProtocolLayersType">
130     <xs:sequence>
131         <xs:element name="Layer" type="ProtocolLayerEnum"/>
132         <xs:element name="Protocol" type="xs:string"/>
133     </xs:sequence>
134 </xs:complexType>
135 
136 <xs:complexType name="StateTableType">
137     <xs:sequence>
138         <xs:element name="TableSize" type="xs:integer"/>
139         <xs:element name="TableStrategy" type="TableStrategyEnum"/>
140     </xs:sequence>
141 </xs:complexType>
142 
143 <xs:complexType name="StatefulType">
144     <xs:sequence>
145         <xs:element name="StateTable" type="StateTableType" minOccurs="1" maxOccurs="1"/
                >
146         <xs:element name="StateTimer" type="StateTimerType" minOccurs="1" maxOccurs="
                unbounded"/>
147         <xs:element name="StateRemovePolicy" type="StatefulPolicyEnum" minOccurs="0"
                maxOccurs="unbounded"/>
148         <xs:element name="NoStatePolicy" type="NoStatePolicyEnum" minOccurs="0"
                maxOccurs="unbounded"/>
149     </xs:sequence>
150     <xs:attribute name="Layer" type="ProtocolLayerEnum"/>
151     <xs:attribute name="Protocol" type="xs:string"/>
152 </xs:complexType>
153 
154 <xs:complexType name="StateTimerType">
155     <xs:sequence>
156         <xs:element name="Timer" type="TimerNameEnum"/>
157         <xs:element name="Value" type="xs:double"/>
158     </xs:sequence>
159 </xs:complexType>
160 
161 <xs:complexType name="FilteringType">
162     <xs:sequence>
163         <xs:element name="State-based" type="FilteringTypeEnum" minOccurs="0" maxOccurs=
                "1"/>
164         <xs:element name="Protocol-based" type="FilteringProtocolBasedEnum" minOccurs="0
                " maxOccurs="1"/>
```

```
165         <xs:element name="Policy-based" type="xs:string" minOccurs="0" maxOccurs="
                unbounded"/>
166     </xs:sequence>
167     <xs:attribute name="Layer" type="ProtocolLayerEnum"/>
168     <xs:attribute name="Protocol" type="xs:string"/>
169 </xs:complexType>
170
171
172 <xs:complexType name="TranslationFieldsType">
173     <xs:sequence>
174         <xs:element name="Mechanism" type="FilteringTypeEnum"/>
175     </xs:sequence>
176 </xs:complexType>
177
178 <xs:complexType name="MappingType">
179     <xs:sequence>
180         <xs:element name="Algorithm" type="MappingAlgorithmEnum"/>
181     </xs:sequence>
182 </xs:complexType>
183
184 <xs:complexType name="BindingType">
185         <xs:sequence>
186                 <xs:element name="PortBinding" type="PortBindingEnum"/>
187                 <xs:element name="NATBinding" type="NATBindingEnum"/>
188                 <xs:element name="AddressPooling" type="PoolingEnum"/>
189         </xs:sequence>
190 </xs:complexType>
191
192 <xs:complexType name="TranslationType">
193         <xs:sequence>
194                 <xs:element name="Mapping" type="MappingType" minOccurs="0" maxOccurs="1"
                        />
195                 <xs:element name="Binding" type="BindingType" minOccurs="0" maxOccurs="1"
                        />
196                 <xs:element name="Fields" type="xs:string" minOccurs="0" maxOccurs="
                        unbounded"/>
197         </xs:sequence>
198         <xs:attribute name="Layer" type="ProtocolLayerEnum"/>
199         <xs:attribute name="Protocol" type="xs:string"/>
200 </xs:complexType>
201
202 <xs:element name="Middlebox">
203    <xs:complexType>
204       <xs:sequence>
205          <xs:element name="NetworkInterface" type="NetworkInterfaceType" minOccurs="0"
                maxOccurs="unbounded"/>
206          <xs:element name="ProtocolLayers" type="ProtocolLayersType" minOccurs="1"
                maxOccurs="unbounded"/>
207          <xs:element name="Stateful" type="StatefulType" minOccurs="0" maxOccurs="
                unbounded"/>
208          <xs:element name="Filtering" type="FilteringType" minOccurs="0" maxOccurs="
                unbounded"/>
209          <xs:element name="Translation_and_Modification" type="TranslationType"
                minOccurs="0" maxOccurs="unbounded"/>
210       </xs:sequence>
211       <xs:attribute name="MiddleboxType" type="xs:string" use="required"/>
212    </xs:complexType>
213 </xs:element>
214
215 </xs:schema>
```

B. LIST OF MIDDLEBOXES

B.1 List of Recommended Home Routers

The following list was extracted from our testset 2 and contains home routers that could be identified by their UPnP data. Of course, there are other models and manufacturers that also showed "good" behavior, but as we could not clearly identify them using UPnP, we cannot list them. We therefore recommend the following models and consider them to behave "good" because:

- they implement an independent binding for UDP and TCP,
- all of them implement port preservation,
- all traversal algorithms for UDP and TCP have been tested successfully.

Manufacturer	**UPnP Model**	**STUN**
ASUSTeK Inc.	ASUS Wireless Router	AR
ASUSTeK Inc.	RT-N66U	PAR
ASUSTek Inc.	RT-N56U Router	PAR
AVM Berlin	FRITZ!Box 2170	PAR
AVM Berlin	FRITZ!Box 6360 Cable (kdg)	PAR
AVM Berlin	FRITZ!Box 7330	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7112 (UI)	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7170 (UI)	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7170	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7240 (UI)	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7270 v1 (UI)	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7270 v2	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7270 v3	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7320 (UI)	AR
AVM Berlin	FRITZ!Box Fon WLAN 7320 (UI)	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7320	PAR
AVM Berlin	FRITZ!Box Fon WLAN 7390	PAR
AVM Berlin	FRITZ!Box WLAN 3170	PAR
AVM Berlin	FRITZ!Box WLAN 3270 v3	PAR
AVM	FRITZ!Box Fon WLAN (UI)	PAR
Ambit	EVW320B	PAR
Buffalo Inc.	Gateway	PAR
Cisco Systems.	Cisco VPN Router	PAR
Cisco	DPC2325	PAR
Cisco	DPC2420	PAR
Cisco	EPC3825	PAR
Cisco	EPC3925	PAR
Cisco	Linksys by Cisco Internet Gateway Device	FC
Cisco	Linksys by Cisco Internet Gateway Device	PAR
D-Link Systems	Wireless N Quadband Router	PAR

Manufacturer	UPnP Model	STUN
D-Link Systems	Wireless N Router	AR
D-Link	D-Link DIR-320	PAR
D-Link	D-Link DIR-615	PAR
D-Link	DIR-655	PAR
D-Link	Wireless Broadband Router	PAR
DD-WRT	Gateway	PAR
Freebox	NASModem/Routeur ADSLFTTH	PAR
Linksys Inc.	WRT54G2	PAR
Linksys	10100 4-Port VPN Router	PAR
Motorola Corporation	SBG6580	PAR
NETGEAR	NETGEAR DGN3500B_16M Router	PAR
NETGEAR	CG3100D	PAR
NETGEAR	CG3101D	PAR
NETGEAR	VMDG480	PAR
NETGEAR, Inc.	DG834Gv5 54 Mbps Wireless Router	AR
NETGEAR, Inc.	DGND3700 RangeMax NEXT	PAR
NETGEAR, Inc.	WNDR4500 N900 Wireless Router	FC
NETGEAR, Inc.	WNR2000 N300 Wireless Router	FC
NETGEAR, Inc.	Wireless-G Router	AR
Netgear	CG814WG v2	PAR
Netgear	CGD24G-100NAS	FC
Netgear	VMDG280	PAR
OpenWRT	OpenWRT router	PAR
Pace plc	Residential Gateway	PAR
SMCD3GNV	SMCD3GNV router	PAR
SmoothWall Express	SmoothWall Express router	PAR
TP-LINK	Wireless N Router TL-WR841N	FC
Technicolor	IGD with UPnP support	AR
Thomson	TWG870U	FC
ZTE	ZTE,pl,step3-sip-fr	PAR
ZyXEL NBG5715	ZyXEL NBG5715 router	PAR

B.2 List of Non-Recommended Home Routers

The following middleboxes that have been identified via UPnP showed a connection dependent behavior for UDP and TCP and none of our traversal tests was successful.

Manufacturer	UPnP Model	STUN
2004W-S	2004W-S	SYM
ASUSTek	ASUS Wireless Router	SYM
AVM Berlin	FRITZ!Box Fon WLAN 7570 vDSL	SYM
Cisco	Internet Access Server	SYM
D-Link	D-Link Corporation UPnP IGD in ISOS 1.00.08.dm12	SYM
D-Link	Xtreme N GIGABIT Router	SYM
D-Link	D-Link DIR-300	SYM
D-Link	D-Link DIR-600	SYM
D-Link	Wireless Gaming Router	SYM
D-Link	Wireless N Router	SYM
FreeBSD	FreeBSD router	SYM
Huawei	Huawei Gateway	SYM
Linksys Inc.	Internet Access Server	SYM
Motorola	Netopia SmartModem	SYM
NETGEAR, Inc.	Linux router	SYM
NETGEAR, Inc.	NETGEAR 802.11 Broadcom Reference RangeMax NEXT	SYM
NETGEAR, Inc.	NETGEAR DGN2200 RangeMax NEXT	SYM
NETGEAR, Inc.	NETGEAR RangeMax N300 Wireless Router WNR2200	SYM
NETGEAR, Inc.	NETGEAR WNDR3400 N600 Wireless Router	SYM
NETGEAR, Inc.	NETGEAR WNDR4000 N750 Wireless Dual Band Gigabit Router	SYM
NETGEAR, Inc.	WNR2000 router	SYM
Netopia, Inc.	Netopia SmartModem	SYM
THOMSON	DSL Internet Gateway Device	SYM
ZyXEL	ZyXEL ZyWALL 10W Internet Security Gateway	SYM

C. NOMADS DOCUMENTATION

C.1 NOMADS Decision Tree

The decision tree as depicted in figure C.1 refers to the pairing procedure for hole punching as described in section 6.4.6. The first part determines the predictability of public endpoints (service S and requester R), whereas the second part makes sure that the TTL field of the hole punching packet is set in a way that a possible answer (to the hole punching packet) as sent by the requester does not close the created mapping by accident.

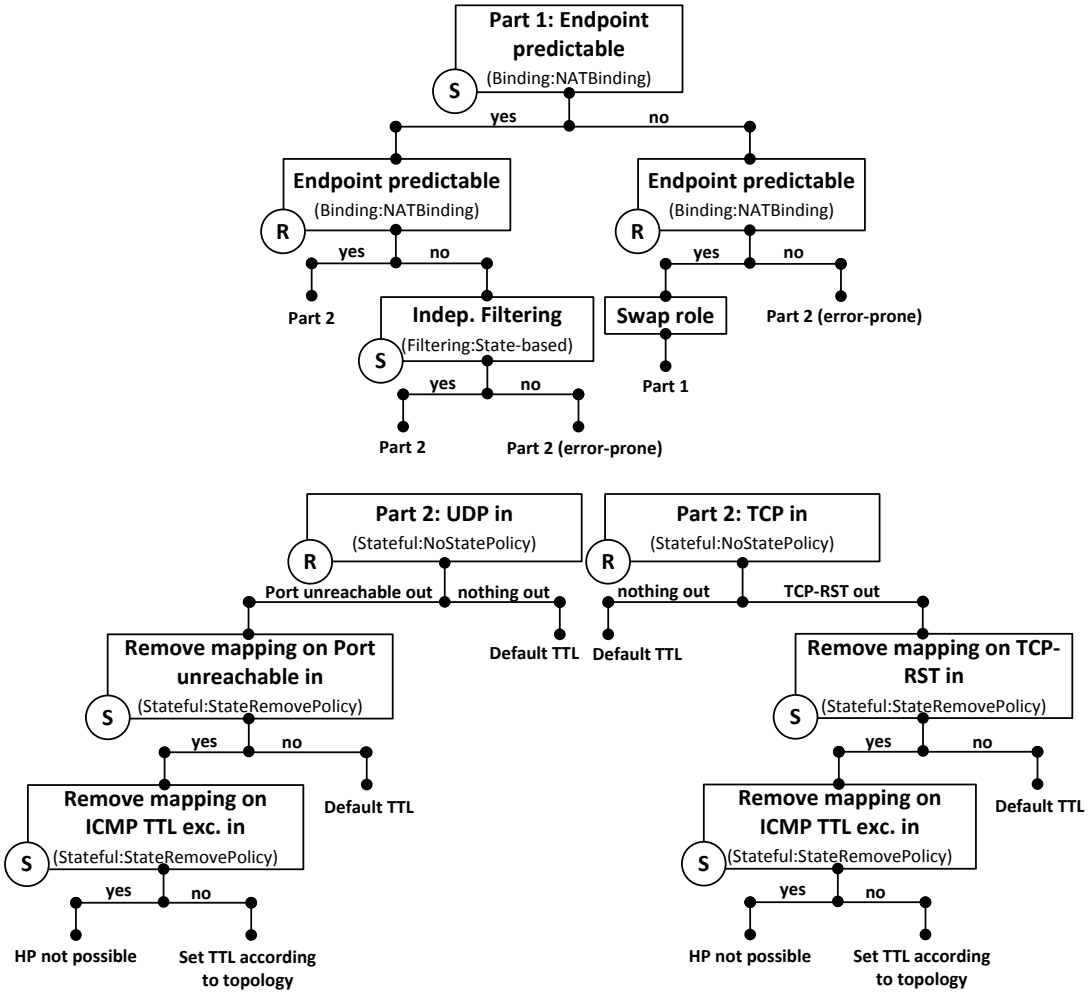

Fig. C.1: Decision tree for hole punching

C.2 NOMADS Signaling Protocol

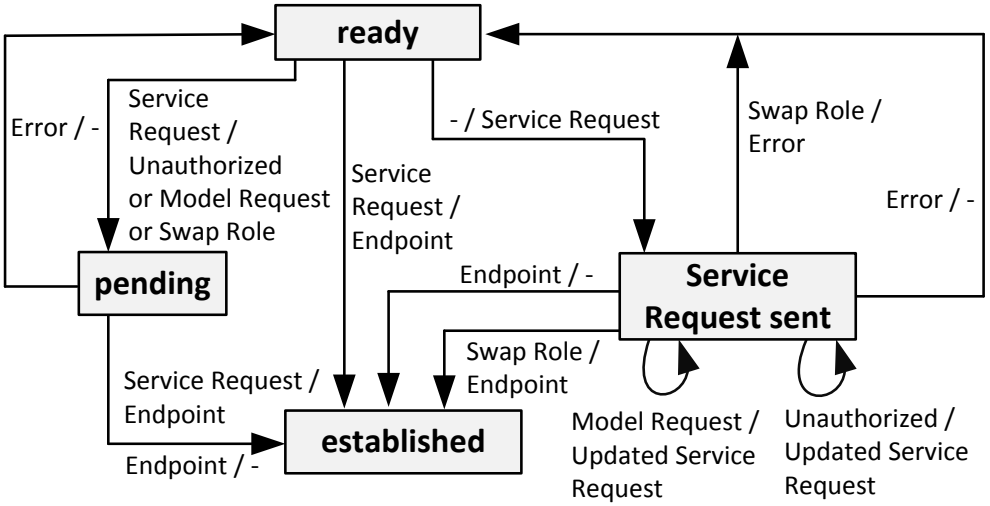

Fig. C.2: State diagram for NOMADS

The state diagram as shown in figure C.2 depicts the individual messages of the request response protocol. The notation we use here is "Input Message / Output Message". "Established" means that the framework reached a state where it is able to forward the initial held back packet (e.g. the first TCP-SYN) to the received endpoint. From now on all packets will use the negotiated endpoint.

C.3 NOMADS Socket API

```
// struct definitions
struct sipaddr
{
  short    sin_family;
  unsigned short sin_port;
  char *   sin_sipaddr;
};

struct sipaddrinfo {
    int ai_flags;
    int ai_family;
    int ai_socktype;
    int ai_protocol;
    size_t ai_addrlen;
    struct sipaddr *sipaddr;
    char *ai_canonname;
    struct addrinfo *ai_next;
};

// create a file descriptor
int nomads_socket(int domain, int type, int protocol);

// connect to the socket given in sipaddr
int nomads_connect(int nomadsfd, const struct sipaddr *addr, socklen_t addrlen);

// send and receive data
```

```
int nomads_send(int nomadsfd, void *msg, int len, int flags);
int nomads_recv(int nomadsfd, void *msg, int len, int flags);

// close connection
int nomads_close(int nomadsfd);

// DNS lookup
// domain -> IP address
int getaddrinfo(const char *node, const char *service, const struct sipaddrinfo *hints,
       struct addrinfo **res);

// IP address -> domain
int getnameinfo(const struct sipaddr *sa, socklen_t salen, char *host, size_t hostlen,
       char *serv, size_t servlen, int flags);

// free struct
void freesipaddrinfo(struct sipaddrinfo *res);
```

List. C.1: The NOMADS socket API

Bibliography

[1] 3GPP. IP Multimedia Subsystem (IMS); Stage 2. TS 23.228, 3rd Generation Partnership Project (3GPP), Sept. 2008.

[2] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), June 2004. Updated by RFC 5247.

[3] C. Aoun and E. Davies. Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status. RFC 4966 (Informational), July 2007.

[4] F. Audet and C. Jennings. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. RFC 4787 (Best Current Practice), Jan. 2007. Updated by RFC 6888.

[5] M. Bagnulo, P. Matthews, and I. van Beijnum. Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers. RFC 6146 (Proposed Standard), Apr. 2011.

[6] M. Bagnulo, A. Sullivan, P. Matthews, and I. van Beijnum. DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers. RFC 6147 (Proposed Standard), Apr. 2011.

[7] F. Baker, X. Li, C. Bao, and K. Yin. Framework for IPv4/IPv6 Translation. RFC 6144 (Informational), Apr. 2011.

[8] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, and X. Li. IPv6 Addressing of IPv4/IPv6 Translators. RFC 6052 (Proposed Standard), Oct. 2010.

[9] D. Barrera, G. Wurster, and P. van Oorschot. Back to the Future: Revisiting IPv6 Privacy Extensions. *LOGIN: The USENIX Magazine*, 36(1):16–26, 2011.

[10] S. Bendeich. A privacy aware Middlebox for IPv6. Bachelor thesis, Technische Universität München, Germany, September 2012.

[11] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig. NATBLASTER: Establishing TCP connections between hosts behind NATs. In *ACM SIGCOMM Asia Workshop, Beijing, China*, 2005.

[12] M. Borella, D. Grabelsky, J. Lo, and K. Taniguchi. Realm Specific IP: Protocol Specification. RFC 3103 (Experimental), Oct. 2001.

[13] M. Borella, J. Lo, D. Grabelsky, and G. Montenegro. Realm Specific IP: Framework. RFC 3102 (Experimental), Oct. 2001.

[14] R. Braden, D. Borman, and C. Partridge. Computing the Internet checksum. RFC 1071, Sept. 1988. Updated by RFC 1141.

[15] B. Carpenter. Architectural Principles of the Internet. RFC 1958 (Informational), June 1996. Updated by RFC 3439.

[16] B. Carpenter. Internet Transparency. RFC 2775 (Informational), Feb. 2000.

[17] B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234 (Informational), Feb. 2002.

[18] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, Feb. 1981.

[19] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927 (Proposed Standard), May 2005.

[20] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. RFC 6763 (Proposed Standard), Feb. 2013.

[21] S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762 (Proposed Standard), Feb. 2013.

[22] S. Cheshire and M. Krochmal. NAT Port Mapping Protocol (NAT-PMP). RFC 6886 (Informational), Apr. 2013.

[23] R. Chinnici, J.-J. Moreau, et al. Web Services Description Language (WSDL) Version 2.0. W3C Recommendation, June 2007.

[24] D. Chouinard, J. Richardson, and M. Khare. H.323 and Firewalls. ITU-T SG16 H.323, Intel White Paper, 2005.

[25] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), Jan. 2008.

[26] D. Clark. The design philosophy of the DARPA internet protocols. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 106–114, New York, NY, USA, 1988. ACM.

[27] D. Clark, L. Chapin, V. Cerf, R. Braden, and R. Hobby. Towards the Future Internet Architecture. RFC 1287 (Informational), Dec. 1991.

[28] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. Updated by RFC 6818.

[29] J. Crowcroft. Net neutrality: The technical side of the debate: a White Paper. *Computer Communication Review*, 37(1):49–56, 2007.

[30] L. D'Acunto, J. Pouwelse, and H. Sips. A measurement of NAT and firewall characteristics in peer-to-peer systems. In *Proc. 15-th ASCI Conference*, pages 1–5. Advanced School for Computing and Imaging (ASCI), June 2009.

[31] D. Degel. NAT-Traversal with DPWS. Bachelor thesis, Technische Universität München, Germany, October 2009.

[32] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson. Fathom: A Browser-based Network Measurement Platform. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, Boston, MA, USA, November 2012.

[33] L. DiCioccio, R. Teixeira, M. May, and C. Kreibich. Probe and Pray: Using UPnP for Home Network Measurements. In *Proceedings of Passive and Active Measurement Conference (PAM)*, Vienna, Austria, March 2012.

[34] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.

[35] T. Dietrich. DNSSEC Support by Home Routers. Technical report, Federal Office for Information Security, 2010.

[36] M. Dietz and D. Grady. Secbook: Formalizing the Facebook Web of Trust. `http://apps.facebook.com/secbook/`, 2011. (Accessed: 04/29/2013).

[37] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[38] R. Dingledine and S. J. Murdoch. Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it. `https://www.torproject.org/press/presskit/2009-03-11-performance.pdf`. (Accessed: 04/29/2013).

[39] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, Dec. 2006.

[40] D. Dolev and A. C. Yao. On the security of public key protocols. Technical report, Stanford University, Stanford, CA, USA, 1981.

[41] M. Dorner. Combining Trusted Computing and Smart Cards for trustworthy VPN access. Bachelor thesis, Technische Universität München, Germany, December 2012.

[42] D. Driscoll and A. Mensch. DPWS Version 1.1. OASIS Standard Specification, July 2009.

[43] A. Durand, R. Droms, J. Woodyatt, and Y. Lee. Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion. RFC 6333 (Proposed Standard), Aug. 2011.

[44] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[45] C. Ellison. DeviceSecurity:1 Service Template. `http://www.upnp.org/`, 2003. (Accessed: 04/29/2013).

[46] B. Elser, G. Groh, and T. Fuhrmann. Group Management in P2P Networks. In *Proceedings of the 2nd IEEE Workshop on Grid and P2P Systems and Applications*, Zurich, Switzerland, August 2010.

[47] J. Eppinger. TCP Connections for P2P Applications - A Software Approach to Solving the NAT Problem. Technical report, Carnegie Mellon University, Pittsburgh, PA, 2005.

[48] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 13–13, Berkeley, CA, USA, 2005. USENIX Association.

[49] M. Ford, M. Boucadair, A. Durand, P. Levis, and P. Roberts. Issues with IP Address Sharing. RFC 6269 (Informational), June 2011.

[50] U. Forum. Internet Gateway Device (IGD) standardized device control protocol, November 2001.

[51] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[52] V. Fuller, T. Li, J. Yu, and K. Varadhan. Supernetting: an Address Assignment and Aggregation Strategy. RFC 1338 (Informational), June 1992. Obsoleted by RFC 1519.

[53] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519 (Proposed Standard), Sept. 1993. Obsoleted by RFC 4632.

[54] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Syst. J.*, 42(1):5–18, Jan. 2003.

[55] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo. IP over P2P: Enabling Self-configuring Virtual IP Networks for Grid Computing. In *Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS-2006*, pages 1–10, 2006.

[56] D. Garcia. Universal plug and play (UPnP) Mapping Attacks. Presented at DEFCON 19, Las Vegas, NV, USA, 2011.

[57] M. Gudgin et al. SOAP Version 1.2. W3C Recommendation, April 2007.

[58] M. Gudgin, M. Hadley, and T. Rogers. Web Services Addressing 1.0 - Core. W3C Recommendation, May 2006.

[59] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh. NAT Behavioral Requirements for TCP. RFC 5382 (Best Current Practice), Oct. 2008.

[60] S. Guha and P. Francis. Characterization and measurement of TCP traversal through NATs and firewalls. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005.

[61] S. Guha and P. Francis. An end-middle-end approach to connection establishment. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 193–204, New York, NY, USA, 2007. ACM.

[62] S. Guha and P. Francis. Identity trail: covert surveillance using DNS. In *Proceedings of the 7th international conference on Privacy enhancing technologies*, PET'07, pages 153–166, Berlin, Heidelberg, 2007. Springer-Verlag.

[63] S. Guha, Y. Takeda, and P. Francis. NUTSS: A SIP based approach to UDP and TCP connectivity. In *Proceedings of the 2004 ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, 2004.

[64] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch. Next Steps in Signaling (NSIS): Framework. RFC 4080 (Informational), June 2005.

[65] M. Handley. Network Neutrality and the IETF. Plenary presentation at the Stockholm IETF meeting, `http://www0.cs.ucl.ac.uk/staff/m.handley/slides/net-neutrality.pdf`, July 2009. (Accessed: 04/29/2013).

[66] M. Handley. Flow processing and the rise of the middle. Presented at MSN 2011, `http://www0.cs.ucl.ac.uk/staff/m.handley/slides/change-coseners2011-print.pdf`, July 2011. (Accessed: 04/29/2013).

[67] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[68] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo. An Experimental Study of Home Gateway Characteristics. In *Proceedings of ACM SIGCOMM Internet Measurement Conference (IMC)*, Melbourne, Australia, November 2010.

[69] T. Heck. An extension for TURN supporting multiple Service Categories. Bachelor thesis, Technische Universität München, Germany, October 2009.

[70] T. Heer, R. Hummen, M. Komu, S. Götz, and K. Wehrle. End-host Authentication and Authorization for Middleboxes based on a Cryptographic Namespace. In *Proceedings of the IEEE International Conference on Communications 2009 (ICC 2009)*, Dresden, Germany, 2009. IEEE.

[71] M. Holdrege and P. Srisuresh. Protocol Complications with the IP Network Address Translator. RFC 3027 (Informational), Jan. 2001.

[72] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape: a thorough analysis of the X.509 PKI using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 427–444, New York, NY, USA, 2011. ACM.

[73] I. Hkan, J.P. Morgan (Goldman Sachs). Nothing But Net: 2011 Internet Investment Guide, January 2011.

[74] IEEE Standards Association. Guidelines for 64-bit global identifier (EUI-64). `http://standards.ieee.org/develop/regauth/tut/eui64.pdf`. (Accessed: 04/29/2013).

[75] International Telecommunications Unions (ITU). The World in 2011: ITC Facts and Figures. `http://www.itu.int/ITU-D/ict/facts/2011/material/ICTFactsFigures2011.pdf`, 2011. (Accessed: 04/29/2013).

[76] C. Jennings. NAT Classification Test Results. Internet Draft - expired, Internet Engineering Task Force, July 2007.

[77] C. Jennings, R. Mahy, and F. Audet. Managing Client-Initiated Connections in the Session Initiation Protocol (SIP). RFC 5626 (Proposed Standard), Oct. 2009.

[78] S. Jiang, D. Guo, and B. Carpenter. An Incremental Carrier-Grade NAT (CGN) for IPv6 Transition. RFC 6264 (Informational), June 2011.

[79] M. E. Jobst. Security and Privacy for Virtual Machine Images in Distributed Environments using Smart Cards. Bachelor thesis, Technische Universität München, Germany, August 2012.

[80] D. Joseph and I. Stoica. Modeling Middleboxes. *IEEE Network Special Issue on Implications and Control of Middleboxes in the Internet*, 22(5):20–25, October 2008.

[81] M. Karsten, S. Keshav, S. Prasad, and M. Beg. An axiomatic basis for Communication. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 217–228, New York, NY, USA, 2007.

[82] J. Kempf, R. Austein, and IAB. The Rise of the Middle and the Future of End-to-End: Reflections on the Evolution of the Internet Architecture. RFC 3724 (Informational), Mar. 2004.

[83] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005. Updated by RFC 6040.

[84] A. Knutsen and A. Ramaiah. TCP option for transparent Middlebox discovery. Internet Draft - work in progress, Internet Engineering Task Force, February 2012.

[85] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), Mar. 2006. Updated by RFCs 5595, 5596, 6335, 6773.

[86] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.

[87] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating The Edge Network. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, Melbourne, Australia, November 2010.

[88] D. Kristol and L. Montulli. HTTP State Management Mechanism. RFC 2965 (Historic), Oct. 2000. Obsoleted by RFC 6265.

[89] Y.-k. Lee, D. Lee, J.-w. Han, and K.-i. Chung. Home Network Device Authentication: Device Authentication Framework and Device Certificate Profile. In *Advances in Web and Network Technologies, and Information Management*, volume 4537 of *Lecture Notes in Computer Science*, pages 573–582. Springer Berlin / Heidelberg, 2007.

[90] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928 (Proposed Standard), Mar. 1996.

[91] L. Lessig. Innovation, Regulation, and the Internet. *The American Prospect*, December 2001.

[92] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), Apr. 2010.

[93] A. Malhotra, K. Warr, D. Davis, and W. Chou. Web Services Eventing (WS-Eventing). W3C Recommendation, December 2011.

[94] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[95] E. Mobile, P. Ab, C. G. Kaisa, K. Nyberg, and C. J. Mitchell. The personal CA – PKI for a Personal Area Network. Technical report, Ericsson Mobile Platforms AB and Nokia Group and University of London, 2002.

[96] V. Modi and D. Kemp. Web Services Dynamic Discovery (WS-Discovery). OASIS Standard Specification, July 2009.

[97] T. Moses. XACML Version 2.0. OASIS Standard Specification, February 2005.

[98] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol. RFC 5201 (Experimental), Apr. 2008. Updated by RFC 6253.

[99] A. Müller. ANTS: An Advanced NAT-Traversal Service for future Home Networks. Diploma thesis, University of Tübingen, Germany, December 2007.

[100] A. Müller, N. Evans, C. Grothoff, and S. Kamkar. Autonomous NAT Traversal. In *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2010)*, Delft, The Netherlands, August 2010.

[101] A. Müller, H. Kinkelin, S. K. Ghai, and G. Carle. An Assisted Device Registration and Service Access System for future Home Networks. In *IFIP Wireless Days 2009*, Paris, France, December 2009.

[102] A. Müller, H. Kinkelin, S. K. Ghai, and G. Carle. A Secure Service Infrastructure for Interconnecting Future Home Networks based on DPWS and XACML. In *HomeNets: ACM SIGCOMM Workshop on Home Networks*, New Delhi, India, September 2010.

[103] A. Müller, A. Klenk, and G. Carle. Behavior and Classification of NAT devices and implications for NAT Traversal. *IEEE Network Special Issue on Implications and Control of Middleboxes in the Internet*, 22(5):14–19, October 2008.

[104] A. Müller, A. Klenk, and G. Carle. On the Applicability of knowledge-based NAT Traversal for future Home Networks. In *Proceedings of IFIP Networking 2008*, Singapore, May 2008.

[105] A. Müller, A. Klenk, and G. Carle. ANTS - A Framework for Knowledge based NAT Traversal. In *IEEE Globecom 2009 Next-Generation Networking and Internet Symposium*, Honolulu, Hawaii, USA, November 2009.

[106] A. Müller, G. Münz, and G. Carle. Collecting Router Information for Error Diagnosis and Troubleshooting in Home Networks. In *Workshop WISE, held in conjunction with the IEEE Conference on Local Computer Networks 2011 (LCN)*, Bonn, Germany, October 2011.

[107] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web Services Security 1.1. OASIS Standard Specification, February 2006.

[108] G. Nalepa. A Unified Firewall Model for Web Security. In *Advances in Intelligent Web Mastering*, volume 43 of *Advances in Soft Computing*, pages 248–253. Springer Berlin / Heidelberg, 2007.

[109] T. Narten, R. Draves, and S. Krishnan. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941 (Draft Standard), Sept. 2007.

[110] B. Nguyen. UPnP RemoteAccess. `http://www.upnp.org/specs/ra/`, 2009. (Accessed: 04/29/2013).

[111] K. Ono and S. Tachimoto. Requirements for End-to-Middle Security for the Session Initiation Protocol (SIP). RFC 4189 (Informational), Oct. 2005.

[112] M.-O. Pahl, C. Niedermeier, M. Schuster, A. Müller, and G. Carle. Knowledge-based middleware for future home networks. In *IFIP Wireless Days 2009*, Paris, France, December 2009.

[113] A. Pastor. Exploiting IGDs remotely via UPnP. `http://www.gnucitizen.org/blog/bt-home-flub-pwnin-the-bt-home-hub-5/`, January 2009. (Accessed: 04/29/2013).

[114] S. Perreault, I. Yamagata, S. Miyakawa, A. Nakagawa, and H. Ashida. Common Requirements for Carrier-Grade NATs (CGNs). RFC 6888 (Best Current Practice), Apr. 2013.

[115] A. Perrig and D. Song. Hash visualization: a new technique to improve real-world security. In *In International Workshop on Cryptographic Techniques and E-Commerce*, pages 131–138, 1999.

[116] J. Peterson. Session Initiation Protocol (SIP) Authenticated Identity Body (AIB) Format. RFC 3893 (Proposed Standard), Sept. 2004.

[117] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826 (INTERNET STANDARD), Nov. 1982. Updated by RFCs 5227, 5494.

[118] J. Postel. Internet Control Message Protocol. RFC 792 (INTERNET STANDARD), Sept. 1981. Updated by RFCs 950, 4884, 6633, 6918.

[119] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), Sept. 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[120] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (INTERNET STANDARD), Oct. 1985. Updated by RFCs 2228, 2640, 2773, 3659, 5797.

[121] J. Quittek, M. Stiemerling, and P. Srisuresh. Definitions of Managed Objects for Middlebox Communication. RFC 5190 (Proposed Standard), Mar. 2008.

[122] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851 (Proposed Standard), July 2004. Obsoleted by RFC 5751.

[123] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), Feb. 1996. Updated by RFC 6761.

[124] E. Rescorla and N. Modadugu. Datagram Transport Layer Security. RFC 4347 (Proposed Standard), Apr. 2006. Obsoleted by RFC 6347, updated by RFC 5746.

[125] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard), June 2000. Updated by RFCs 2868, 3575, 5080, 6929.

[126] A. Rijsinghani. Computation of the Internet Checksum via Incremental Update. RFC 1624 (Informational), May 1994.

[127] J. Roberts. The clean-slate Approach to future Internet Design: a Survey of Research Initiatives. *Annals of Telecommunications*, 64:271–276, 2009.

[128] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard), Apr. 2010. Updated by RFC 6336.

[129] J. Rosenberg, A. Keranen, B. B. Lowekamp, and A. B. Roach. TCP Candidates with Interactive Connectivity Establishment (ICE). RFC 6544 (Proposed Standard), Mar. 2012.

[130] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), Oct. 2008.

[131] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878.

[132] J. Rosenberg and H. Tschofenig. Discovering, Querying, and Controlling Firewalls and NATs. Internet Draft - expired, Internet Engineering Task Force, October 2007.

[133] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), Mar. 2003. Obsoleted by RFC 5389.

[134] E. S. Perreault, I. Yamagata, S. Miyakawa, A. Nakagawa, and H. Ashida. Common requirements for Carrier Grade NATs (CGNs). Internet Draft - work in progress, Internet Engineering Task Force, August 2012.

[135] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), Oct. 2004. Obsoleted by RFC 6120, updated by RFC 6122.

[136] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2:277–288, November 1984.

[137] H. Schulzrinne and R. Hancock. GIST: General Internet Signalling Transport. RFC 5971 (Experimental), Oct. 2010.

[138] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard), Apr. 1998.

[139] S. Son. *Middleware approaches to middlebox traversal*. PhD thesis, University of Wisconsin at Madison, Madison, WI, USA, 2006. AAI3222909.

[140] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), Jan. 2001.

[141] P. Srisuresh, B. Ford, S. Sivakumar, and S. Guha. NAT Behavioral Requirements for ICMP. RFC 5508 (Best Current Practice), Apr. 2009.

[142] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663 (Informational), Aug. 1999.

[143] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan. Middlebox communication architecture and framework. RFC 3303 (Informational), Aug. 2002.

[144] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), Sept. 2007. Updated by RFCs 6096, 6335.

[145] R. Stewart, M. Tuexen, and I. Ruengeler. Stream Control Transmission Protocol (SCTP) Network Address Translation. Internet Draft - work in progress, Internet Engineering Task Force, February 2013.

[146] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960 (Proposed Standard), Oct. 2000. Obsoleted by RFC 4960, updated by RFC 3309.

[147] M. Stiemerling, H. Tschofenig, C. Aoun, and E. Davies. NAT/Firewall NSIS Signaling Layer Protocol (NSLP). RFC 5973 (Experimental), Oct. 2010.

[148] The Monkeysphere Project. Monkeysphere. `http://web.monkeysphere.inf/`, 2011. (Accessed: 04/29/2013).

[149] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures Second Edition. W3C Recommendation, October 2004.

[150] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), Sept. 2007.

[151] K. Tobe, A. Shimoda, and S. Goto. Extended UDP Multiple Hole Punching Method to Traverse Large Scale NAT. In *Proceedings of the 30th Asia Pacific Advanced Network Meeting*, Hanoi, Vietnam, August 2010.

[152] Trusted Computing Group. Architecture Overview, Rev. 1.4. TCG Specification, 2007.

[153] G. Tsirtsis and P. Srisuresh. Network Address Translation - Protocol Translation (NAT-PT). RFC 2766 (Historic), Feb. 2000. Obsoleted by RFC 4966, updated by RFC 3152.

[154] M. Tuxen, I. Rungeler, R. Stewart, and E. Rathgeb. Network Address Translation for the Stream Control Transmission Protocol. *IEEE Network Special Issue on Implications and Control of Middleboxes in the Internet*, 22(5):26–32, October 2008.

[155] A. Ulrich, R. Holz, P. Hauck, and G. Carle. Investigating the OpenPGP Web of Trust. In *Computer Security ESORICS 2011*, volume 6879 of *Lecture Notes in Computer Science*, pages 489–507. Springer Berlin / Heidelberg, 2011.

[156] B. van Schewick. The Network Neutrality Debate - An Overview. Talk at the Technical Plenary of IETF 75, `http://www.ietf.org/proceedings/75/slides/plenaryt-5.pdf`, July 2009. (Accessed: 04/29/2013).

[157] Z. Wang and J. Crowcroft. A Two-Tier Address Structure for the Internet: A Solution to the Problem of Address Space Exhaustion. RFC 1335 (Informational), May 1992.

[158] Z. Wang, Z. Qian, Q. Xu, Z. M. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, Toronto, Ontario, Canada, 2011.

[159] M. Wasserman and F. Baker. IPv6-to-IPv6 Network Prefix Translation. RFC 6296 (Experimental), June 2011.

[160] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger. IANA-Reserved IPv4 Prefix for Shared Address Space. RFC 6598 (Best Current Practice), Apr. 2012.

[161] B. Wellington. Domain Name System Security (DNSSEC) Signing Authority. RFC 3008 (Proposed Standard), Nov. 2000. Obsoleted by RFCs 4035, 4033, 4034, updated by RFC 3658.

[162] D. Wing, S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk. Port Control Protocol (PCP). RFC 6887 (Proposed Standard), Apr. 2013.

[163] F. Wohlfart. Topology-based Traversal of Large Scale NAT. Masters thesis, Technische Universität München, Germany, July 2012.

[164] D. I. Wolinsky. *On the design and implementation of user-friendly, self-configuring, and scalable Virtual Private Networks.* PhD thesis, University of Florida, Gainsville, FL, USA, 2009.

[165] D. I. Wolinsky, K. Lee, P. O. Boykin, and R. Figueiredo. On the Design of Autonomic, Decentralized VPNs. In *Proceedings of the International Conference on Collaborative Computing (CollaborateCom)*, Chicago, IL, USA, October 2010.

[166] R. Yavatkar, D. Pendarakis, and R. Guerin. A Framework for Policy-based Admission Control. RFC 2753 (Informational), Jan. 2000.

[167] L. Zhang. A retrospective View of Network Address Translation. *IEEE Network Special Issue on Implications and Control of Middleboxes in the Internet*, 22(5):8–12, October 2008.

[168] L. Zhang, W. Jia, X. Xiao, B. Dai, and H. Li. Research of TCP NAT Traversal Solution Based on Port Correlation Analysis & Prediction Algorithm. In *International Conference on Wireless Communications, Networking and Mobile Computing*, 2010.

ISBN 3-937201-35-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
DOI: 10.2313/NET-2013-07-1