Lehrstuhl für Integrierte Schaltungen
Technische Universität München

Speculative Protocol-Processing for High-Speed Packet Forwarding

Jürgen Foag

Lehrstuhl für Integrierte Schaltungen
Technische Universität München

Speculative Protocol-Processing for High-Speed Packet Forwarding

Jürgen Foag

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. Jörg Eberspächer
Prüfer der Dissertation:
1. Univ.-Prof. Dr.-Ing. Ingolf Ruge, em.
2. Univ.-Prof. Dr.-Ing. Erik Maehle, Universität zu Lübeck
3. Univ.-Prof. Dr. sc. techn. (ETH) Andreas Herkersdorf

Die Dissertation wurde am 18.09.2003 bei der Technischen Universität München
eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik
am 26.02.2004 angenommen.

Rien n’est plus fort qu’une idée dont l’heure est venue !
(Victor Hugo)

To Lydia, Mareike, Anyesse and everyone who will follow ...

with all my love !

Acknowledgments

I would like to express my sincere appreciation and gratitude to my supervisor, Prof. Ingolf Ruge, chair of the Institute for Integrated Circuits at the Technical University of Munich. He encouraged me in research of integrated circuits in the application field of High-Speed Networking.

I am also indebted to Prof. Erik Maehle and Prof. Andreas Herkersdorf for accepting to act as competent co-referees for this dissertation. Their insights and constructive criticism inspire for further activities in the challenging area of network processors.

Thanks also to Prof. Eberspächer for serving as the chair for the oral defense of my dissertation.

Likewise, this work drew inspiration and invaluable support from Dr. Thomas Wild and Dr. Walter Stechele. They have listened patiently my ideas of speculative protocol processing.

This work also benefited from discussions with Dr. Oliver Denk, Gregory Gagarin, Dr. Iyad Kanj, Oliver Laub, Dr. Damien Magoni, Osnat Mokryn and Maresa Praxenthaler.

I would like to also thank my colleagues at the Institute for Integrated Circuits, namely Stephan Herrmann, Wolfgang Kohtz, Dr. Torsten Mahnke, Dr. Nuria Pazos, Stephan Stilkerich and Armin Windschiegl.

This work was influenced in particular by experiences I have made in the S/390 Microprocessor Development of IBM Böblingen, Germany. Special thanks are denoted to Ralph Koester.

This work would never have been realized without the love, help, support and female inspirations of my family Lydia, Mareike, Anyesse as well as my sisters Isabell, Monika, Ute and Regina. Finally, I am indebted to my mother Lieselotte and my father Lorenz (†) for all they gave to me during my life.

Kurzfassung

Netzprozessoren sind Bausteine, die in zunehmendem Maße in Internetroutern eingesetzt werden. Neben der flexiblen Unterstützung von Protokollen und Diensten zeichnen sich Netzprozessoren durch eine hohe Leistungsfähigkeit aus. Die Flexibilität wird durch eingebettete Prozessoren mit optimiertem Instruktionssatz realisiert. Eine hohe Performanz wird durch geeignete Systemarchitekturen erzielt, die sowohl Multiprozessierung als auch Multithreading unterstützen. Wesentliche Performanzmetriken bei Netzwerkkomponenten sind Datendurchsatz und Bearbeitungslatenz. Während bisherige Netzprozessoren auf hohe Datendurchsatzwerte optimiert sind, wird die Bearbeitungslatenz von architektureller Seite des Netzprozessors in der Regel vernachlässigt. Dies kann sich bei Echtzeitanwendungen, z.B. Internettelefonie, als nachteilig erweisen. Stattdessen sollen eine beschleunigte Paketweiterleitung und somit verkürzte Latenzen in Netzknoten mittels geeigneter Dienstarchitekturmodelle, wie z.B. *Differentiated Services*, und veränderten Weiterleitungsprinzipien wie *Multiprotocol Label Switching* ermöglicht werden.

Zentraler Bestandteil dieser Arbeit ist eine Protokollbearbeitungsmethode für Netzwerkprozessoren, die auf kurze Verzögerungszeiten bei der Protokollverarbeitung fokussiert ist. Die Methode besteht aus 2 Teilen: mittels *Protokollstapelprädiktion* wird aus einer Menge von in der Vergangenheit empfangenen Paketen der Protokollstapel des als nächstes empfangenen Pakets vorhergesagt. Die Bearbeitung von unterschiedlichen Netzwerkschichten zugehörigen Funktionen dieses Pakets wird durch *spekulative Protokollbearbeitung* gleichzeitig in parallelen Bearbeitungseinheiten begonnen. Im Falle einer korrekten Vorhersage ist die Bearbeitungslatenz verkürzt. Anderenfalls ergeben sich höhere Bearbeitungsdauern. Zusammengefasst wird für die Bearbeitungsdauer eine im statistischem Mittel reduzierte Latenz erwartet.

Im ersten Teil der Arbeit wird eine abstrakte Konzeptevaluierung durchgeführt. Hierzu werden netzwerk- und implementierungsspezifische Merkmale separiert und die das Konzept beeinflussenden Parameter extrahiert. In einzelnen Szenarien wird dann der quantitative Einfluss jeweils eines Parameters auf Latenzreduktion und zusätzlichen Prozessierungsaufwand hin analysiert. Es wird unter anderem gezeigt, dass mittels einer hohen Vorhersagegüte wesentlich die Latenzreduktion erhöht und der zusätzliche Prozessierungsaufwand gesenkt werden kann.

Um das Verhalten der Bearbeitungslatenz im Internet-spezifischen Anwendungsfall zu analysieren, wird ein System spezifiziert, das das spekulative Bearbeitungskonzept implementiert. Hierbei handelt es sich nicht um ein optimiertes Netzwerkprozessorsystem, das mit kommerziell erhältlichen Bausteinen vergleichbar ist. Stattdessen dient es zur Bestimmung der erzielbaren Latenzreduktion unter implementierungsspezifischen Aspekten, z.B. Verwendung einer aufgeteilten Busarchitektur. Da das Konzept eine hohe Latenzreduktion an Einsatzorten mit komplexer Paketbearbeitung verspricht, wird hierfür der Zugangsbereich zu Internetdomänen gewählt. Als Prädiktionsalgorithmus wird ein Verfahren verwendet, das auf dem am häufigsten aufgetretenen Ereignis basiert. Das dynamische Verhalten ermöglicht eine selbstständige Anpassung an veränderte Netzwerkverkehrscharakteristiken. Zum Zweck der schnellen Vorhersagetransienz und zur Vermeidung von möglichen Vorhersageregisterüberläufen ist es um einen zyklischen Rücksetzungsmechanismus erweitert.

Sowohl für die Systemsimulation als auch die Simulation der Vorhersageeinheit werden aktuelle Netzwerkverkehrsstatistiken verwendet. Basierend auf Verkehrsstatistiken von 2002 ergaben sich bei der Simulation der zweistufigen Vorhersageeinheit Raten für korrekte Vorhersagen von 82 % für die erste und ca. 98 % für erste und zweite Vorhersagestufe zusammen. Grundsätzlich läßt sich feststellen, dass die mittlere Zeitdauer zur Berechnung einer neuen Vorhersage deutlich geringer ist als die mittlere Bearbeitungslatenz eines Pakets. Dies ermöglicht eine Aktualisierung der Vorhersagetabellen für jedes Paket. Für die Systemsimulation wird als Referenzimplementierung ein Modell verwendet, dass die Bearbeitung in parallelen Prozessoren pseudo-parallel, d.h. zeitversetzt entsprechend bestehender Datenabhängigkeiten und deren Auflösung, ausführt. Für die exemplarisch gewählte Systemumgebung am Netzwerkedge ergaben sich Werte zwischen 6,4 und 22,5 % für die Latenzreduktionen. Als Nachteil ergibt sich jedoch eine höhere Auslastung der eingebetteten Prozessoren im Fall der spekulativen Bearbeitungsmethode.

Abschliessend wird die Protokollbearbeitungsmethode aus Anwendungssicht betrachtet. Fehlvorhersagen verursachen längere Bearbeitungsdauern und generieren folglich Verzögerungsschwankungen, die sich auf Echtzeitanwendungen wie IP-Telefonie nachteilig auswirken können. Zur Kompensation von Verzögerungsschwankungen wird ein netzwerkknoteninterner Ansatz gemacht, der im Warteschlangensystem implementiert ist. Hierzu wird ein als Referenz gewählter weighted-fair-queuing Algorithmus um zwei Prioritätswarteschlangen erweitert. In diese werden Pakete mit einer bzw. mit zwei Fehlvorhersagen abgelegt. Die Untersuchungen zeigen, dass hierdurch zeitliche Schwankungen bei Paketen, die schnelle Paketweiterleitung erfordern, ausgeglichen werden können. Zuletzt wird die Latenzreduktion des spekulativen Netzprozessors gegenüber pseudoparallelen Referenzimplementierungen auf 13,4 bis 14,9 % abgeschätzt.

Zusammenfassend wird festgestellt, dass das Konzept der Protokollvorhersage und spekulativen Paketbearbeitung gewinnbringend in Routern am Internetrandbereich und zwischen den Domänen von Internetdiensteanbietern eingesetzt werden kann, um deren Bearbeitungsdauern zu verringern.

Résumé

Les processeurs réseaux sont des composants qui sont de plus en plus utilisés dans les routeurs de l'Internet. En plus de fournir un support flexible pour les protocoles et les services, les processeurs réseaux fournissent de hautes performances. La flexibilité est fournie par des processeurs embarqués qui possèdent un jeu d'instruction optimisé. De hautes valeurs de performances sont possibles grâce à des architectures système appropriées, qui supportent les processeurs multiples et les processus légers multiples (multithreading). Les principales métriques de performance des composants réseaux sont le débit et le délai de traitement. Bien que les processeurs réseaux courants soient optimisés pour obtenir de hautes valeurs de débit de données, le délai est couramment négligé du point de vue de l'architecture du processeur réseau. Cette lacune peut être désavantageuse dans le cas d'une application temps-réel, e.g. téléphonie sur IP. A la place, des modèles adéquats d'architecture de services, e.g. *differentiated services* et des principes d'acheminement modifiés tels que *Multi-protocol label switching* sont définis afin d'accélérer l'acheminement des paquets entranant ainsi une diminution des délais dans les noeuds du réseau.

La partie principale de cette thèse est une méthodologie de traitement de protocoles dans les processeurs réseaux qui se focalise sur des délais courts dans le traitement de protocoles. Elle est constituée de deux éléments : la *Prédiction de protocole (Protocol-stack prediction)* utilise en entrée un jeu de paquets précédemment reus et prédit le prochain paquet qui va être reu. L'exécution de fonctionnalités qui se réfèrent à des couches réseaux différentes, i.e. le *Traitement spéculatif de protocoles (speculative protocol-processing)*, est démarré simultanément sur des ressources de traitement parallèles. Le délai de traitement est réduit dans le cas d'une prédiction correcte. Sinon, des valeurs plus grandes seront obtenues. Au total, une moyenne réduite du délai de traitement est attendue.

Une évaluation conceptuelle abstraite est réalisée dans la première partie de cette thèse. Les caractéristiques spécifiques liées au réseau et à l'implémentation sont séparées et les paramètres liés aux concepts immanents sont extraits dans ce but. Leur impact quantitatif sur la réduction du délai et le cot de traitement additionnel est déterminé pour des scénarios individuels. Nous montrons que la réduction du délai peut être augmentée et le surcot de traitement peut être diminué grâce à des valeurs élevées sur la précision de la prédiction.

Afin d'analyser le délai dans un environnement d'application réseau, une spécification du système spéculatif est réalisée. Elle ne peut pas être considérée comme un système processeur réseau optimisé qui permettrait une comparaison avec des appareils disponibles dans le commerce. Cependant elle peut être utilisée pour calculer la réduction de délai réalisable selon certaines considérations liées à des aspects de l'implémentation, par exemple une architecture avec un bus partagé. L'architecture est un modèle hybride qui effectue un traitement parallèle des paquets au niveau du système et un traitement parallèle des en-têtes au niveau de la couche réseau. Le système sera déployé en bordure de réseau car un gain élevé sur le délai est attendu dans les zones qui nécessitent une haute complexité de traitement des paquets.

L'algorithme de prédiction est basé sur le principe du plus fréquemment utilisé. Une adaptation autonome aux modifications des caractéristiques du trafic réseau est produite par le comportement dynamique de la prédiction. Une possibilité additionnelle de réinitialisation cyclique permet d'obtenir une adaptation souple de la prédiction et d'éviter le débordement des registres de prédiction.

Des statistiques de trafic réseau courantes sont utilisées dans la simulation du système et dans celle de l'unité de prédiction. Des taux de succès élevés de 82 % pour la première prédiction seule et approximativement de 98 % pour la première et la deuxième prédictions réunies peuvent être obtenus. Il peut être démontré que le temps de calcul moyen d'une prédiction est significativement inférieur au délai de traitement moyen d'un paquet. Par conséquent, une mise à jour de la table de prédiction peut être réalisée pour chaque paquet à la vitesse de ligne des paquets. Une implémentation de référence qui effectue un traitement de protocoéle d'une manière pseudo parallèle, i.e. le départ est décalé dans le temps selon les dépendances des données, est utilisée pour la simulation du système. Les résultats de simulation concernant la réduction du délai vont de 6,4 à 22,5 %. Cependant la méthode de traitement spéculative entrane une charge de travail plus élevée des processeurs embarqués.

Par la suite, la méthodologie de traitement de protocole est considérée du point de vue des applications. Les prédictions erronées entranent des délais de traitement augmentés et par voie de conséquence produisent une gigue qui affecte les applications temps-réel telles que la téléphonie sur IP. Afin de compenser cette gigue, une approche interne au noeud réseau de l'implémentation du système de file d'attente est proposée. Une implémentation de référence d'un algorithm de file d'attente à poids équilibrés (weighted-fair queuing) est étendue par deux files de priorité dans lesquelles sont insérées les paquets provenant d'une ou deux prédictions erronées. Les études de performance démontrent que les variations temporelles des paquets qui nécessitent un acheminement expéditif peuvent être compensées. Finalement, la réduction du délai de traitement dans le processeur réseau spéculatif est estimée comprise entre 13,4 et 14,9 %.

Nous pouvons en conclure que le concept de prédiction de protocole et de traitement spéculatif de paquet peut être mis à profit dans les routeurs de bordure de réseau et ceux situés entre les domaines des fournisseurs d'accès à l'Internet.

Abstract

Network processors are devices which are increasingly applied in Internet routers. Besides a flexible support of protocols and services, network processors provide high performances. The flexibility is provided by embedded processors which feature an optimized instruction set. High performance values are enabled by appropriate system architectures, which support multiprocessing as well as multithreading. Main performance metrics of networking devices are throughput and processing latency. While current network processors are optimized for high values of the data throughput, the latency is commonly neglected from the perspective of the network processor architecture. This lack can cause disadvantages in case of real-time application, e.g. IP telephony. Instead of that, well-suited service architecture models, e.g. *differentiated services* and modified forwarding principles such as *Multi-protocol label switching* are defined to enable accelerated packet forwarding and consequently reduced latencies in network nodes.

The main part of this thesis is a methodology for protocol-processing in network processor which is focused on short delays for protocol-processing. It consists of two components: *Protocol-stack prediction* uses a set of earlier received packets as input and predicts the packet which will be received next. The execution of functionalities which refer to different network layers, i.e. the *speculative protocol-processing*, is started simultaneously in parallel processing resources. The processing latency is decreased in case of a correct prediction. Otherwise, increased values will be achieved. In total, a reduced mean latency for the processing delay is expected.

An abstract concept evaluation is done in the first part of this thesis. Networking- and implementation-specific characteristics are separated and concept-immanent parameters are extracted for these purposes. Their quantitative impact on the latency reduction and the additional processing effort is determined in individual scenarios. It is shown that the latency reduction can be increased and the additional processing effort can be decreased through high values of the prediction accuracy.

In order to analyze the latency in a networking-specific application environment, a specification of the speculative system is done. It is not aimed as an optimized network processor system which allows a comparison with commercially available devices. Instead, it can be used for a computation of the achievable latency reduction under consideration of implementation-specific aspects. The architecture represents a hybrid model that realizes packet parallelism on system-level and network-layer parallelism on packet-processing element level. The system will be applied at the network edge because of an expectation for a high latency gain in operation areas which require a high packet-processing complexity. The prediction algorithm is based on the principle of most-frequently used. An autonomous adaptation to modified network traffic characteristics is given by the dynamic behavior of the prediction. An additional cyclic reset facility serves to obtain a smooth prediction adaptation and to avoid prediction register overflows.

Current network traffic statistics are used for the system simulation and the simulation of the two-stage prediction unit. Based on network traffic statistics of 2002, hit rates of 82 % for the first and approximatively 98 % for the first and second prediction stage can be achieved. In can be asserted that the mean computation delay for a prediction is significantly less than the mean processing delay of a packet. Thus, an update of the prediction table can be done for each packet in line-rate. A reference implementation which executes protocol-processing in a pseudo-parallel manner, i.e. the initiation of task processing is shifted in time according to data dependencies, is used for the system simulation. If the network edge is used as system environment for the simulation, the results of the latency reduction range between 6.4 and 22.5 %. The speculative processing method, however, leads to a higher workload of the embedded processors.

Subsequently, the protocol-processing methodology is considered from application perspective. Mispredictions cause increased processing delays and consequently generate a delay jitter which affects real-time applications such as IP telephony. In order to compensate this jitter, a network-node internal approach for implementation within the queuing system is proposed. A reference implementation of a weighted-fair queuing algorithm is extended by two priority queues in which packets with one or two occurred mispredictions are inserted. Performed studies demonstrate that time deviations of packets which require expedited forwarding can be compensated. Finally, the latency reduction of the speculative network processor is estimated to 13.4 to 14.9 %.

It can be concluded that the concept of protocol-stack prediction and speculative packet-processing can be inserted profitably in routers at the network edge and between Internet service provider domains.

Contents

1 Introduction **1**

1.1 Internetworking 1

1.1.1 Network architectures 1

1.1.2 Communication reference model and service architectures models . 2

1.1.3 Network performance metrics 7

1.2 Network processors 8

1.3 Motivation 10

1.4 Structure of this thesis 11

2 Protocol-processing methodologies **13**

2.1 Introduction 13

2.2 Packet-processing 14

2.2.1 Packet reception 15

2.2.2 Header parsing & checking 15

2.2.3 Forwarding 16

2.2.4 Classification 16

2.2.5 Policing, metering, shaping 17

2.2.6 Queuing, scheduling, congestion avoidance 17

2.2.7 Packet transmission 18

2.2.8 Additional and optional tasks 18

2.3 Protocol specifications 18

2.3.1 Light-weight protocols 18

2.3.2 Adaptive protocol-stacks 19
2.4 Protocol implementations 20
2.4.1 Integrated-layer processing 20
2.4.2 Connection-oriented protocol-processing optimization 21
2.4.3 Multiprocessing / Parallel protocol-processing 24
2.4.4 Multithreading 27
2.5 Processing flow 33
2.6 Predictive processing models 34
2.6.1 Fundamentals 34
2.6.2 Data prediction 34
2.7 Network Traffic 36
2.8 Summary 38

3 Speculative protocol-processing concept 40
3.1 Introduction 40
3.2 Protocol-stack prediction 41
3.3 Speculative protocol-processing 44
3.4 Concept benefit 48

4 Abstract concept evaluation 49
4.1 Introduction 49
4.2 Evaluation 50
4.2.1 Predefinitions 50
4.2.2 Scenarios 56
4.2.2.1 Scenario 1: Number of predicted layers 56
4.2.2.2 Scenario 2: Additional processing time in case of misprediction 59
4.2.2.3 Scenario 3: Prediction accuracy 60
4.2.2.4 Scenario 4: Resolution time of control-flow conditions . . 63
4.2.2.5 Scenario 5: Processing time of control-flow conditions . . 64
4.2.2.6 Scenario 6: Prediction model: single-level or multi-level . 64
4.3 Conclusion 65

5 System design 68

5.1 Introduction 68

5.2 Constraints of the application example 69

5.2.1 System environment 69

5.2.2 Supported protocols 69

5.2.3 Supported services 70

5.2.4 Protocol-stack 70

5.3 Functional specification 70

5.3.1 Overview 70

5.3.2 Input processing 71

5.3.3 Protocol-stack processing 72

5.3.4 Output processing 74

5.3.5 Control-point processing 76

5.3.6 Protocol-stack prediction 76

5.4 System architecture 84

5.4.1 Overview 85

5.4.2 Packet-processing element architecture 87

5.4.3 Network-layer processing unit and control-point PU 88

5.4.4 Prediction unit 89

6 System evaluation 91

6.1 Introduction 91

6.2 Prediction PU 92

6.2.1 Test methodology and simulation traffic 92

6.2.2 Simulations and analysis 94

6.3 Packet-processing element 103

6.3.1 System implementation 103

6.3.2 Test methodology and constraints 108

6.3.3 Simulations and analysis 109

6.4 Conclusion 116

7 Concept application study 118

7.1 Introduction . . . 118

7.2 Queuing delay and jitter analysis . . . 118

7.2.1 Motivation . . . 118

7.2.2 Scheduling algorithm . . . 119

7.2.3 Evaluation . . . 120

7.3 Network processor delay . . . 125

7.4 End-to-end analysis . . . 126

8 Summary and conclusions 131

Bibliography 135

A Publications and Patent Applications 146

List of Figures

1.1 Internet Topology 2
1.2 Architecture model according to ISO/OSI Reference Model 3
1.3 Differentiated Services Architecture 5
1.4 Multiprotocol Label Switching 6
1.5 Virtual Private Network with IPSec 6
1.6 End-to-end-Latency 7
1.7 Router architecture 9
1.8 Network Processor Block Diagram 10

2.1 Packet Processing Chain 14
2.2 IPv4 packet 16
2.3 Integrated-layer Processing 21
2.4 Jacobsons' header prediction 22
2.5 Woodsides' Protocol Bypassing Scheme 23
2.6 Protocol Bypass Architecture 23
2.7 Horizontal Process Architectures 25
2.8 Vertical Process Architectures 26
2.9 Run-to-completion model 26
2.10 Systolic model 27
2.11 Coarse-grained and Fine-grained Multithreading 28
2.12 Simultaneous Multithreading 29
2.13 2-way Multithreading implementation 30

2.14 2-way MT processing flow 30

2.15 Super-scalar processor implementation 31

2.16 Processing flow of a super-scalar processor implementation 31

2.17 Multithreading Implementation (2 functional processing units) 32

2.18 Processing flow of a MT implementation (2 functional PU) 32

2.19 Sequential processing flow IXP 1200 33

2.20 Program execution in microprocessors 35

3.1 Processing concept 41

3.2 Protocol-stack prediction scheme 42

3.3 Coverage of protocol-stack prediction 43

3.4 Speculative Packet Processing (Layer 2 and 3 hit) 45

3.5 Speculative Packet Processing (Layer 2 miss / Layer 3 hit) 46

3.6 Speculative Packet Processing (Layer 2 and 3 miss) 47

3.7 Sequential and pseudo-parallel processing 47

4.1 Layer hierarchy 50

4.2 Pseudo-parallel processing implementation 51

4.3 Speculative processing / case prediction hit 52

4.4 Speculative processing / case misprediction 52

4.5 Costs in case of misprediction 54

4.6 Analytical model 55

4.7 Processing flows (layer 2 - 3) 56

4.8 Pseudo-parallel processing flow (layer 2 - 4) 57

4.9 Pseudo-parallel processing flow (layer 2 - 5) 57

4.10 Scenario 1: $\overline{AC},\overline{r}$ / Parameter: number of predicted layers 58

4.11 Scenario 1: $\overline{AC},\overline{r}(N_h/N = N_{hh}/N = N_{hhh}/N = 0.5)$ / Parameter: number of predicted layers 58

4.12 Scenario 1: $\overline{AC},\overline{r}(N_h/N = N_{hh}/N = N_{hhh}/N = 0.7)$ / Parameter: number of predicted layers 59

4.13 Scenario 2: $\overline{r} = function(T_{APT})$ 59

4.14 Scenario 2: $\overline{AC}, \overline{r}$ / Parameter: T_{APT} 60
4.15 Scenario 3: $\overline{AC}, \overline{r}$ / Parameter: prediction accuracy 61
4.16 Scenario 3: $\overline{AC}, \overline{r}$ / Parameter: prediction accuracy, $T_{APT} \neq 0$) 62
4.17 Scenario 3: $\overline{AC}, \overline{r}$ / Parameter: prediction accuracy, $T'_{L3_1} = T_{L3_1} + \delta$ 62
4.18 Scenario 4: Completion time of the checktask 63
4.19 Scenario 4: $\overline{AC}, \overline{r}$ / Parameter: resolution time of control-flow conditions . . 63
4.20 Scenario 5: Latency reduction / additional cost 64
4.21 Scenario 5: $\overline{AC}, \overline{r}$ / Parameter: processing time of control-flow conditions . 65
4.22 Single-level prediction (a) and multi-level prediction (b) 66
4.23 Scenario 6: $\overline{AC}, \overline{r}$ / Parameter: prediction model 66

5.1 Supported protocols 69
5.2 PPE processing chain 71
5.3 Input processing flow 72
5.4 Protocol-stack processing flow 73
5.5 Output processing flow 75
5.6 MFU computation flow 78
5.7 Components of protocol-stack prediction 82
5.8 Special register file 83
5.9 Prediction table reset facility 84
5.10 System model 85
5.11 System architecture 86
5.12 Packet Processing Element 87
5.13 Network-layer processing unit 88
5.14 Prediction Processing Unit 89
5.15 Multi-modal protocol-stack prediction 90
5.16 Protocol-stack status word (PSSW) 90

6.1 Prediction PU evaluation test bench 92
6.2 Misprediction rate 96

6.3 Misprediction rate, different weights 97
6.4 Hit rate UDP packets 98
6.5 Traffic 1 100
6.6 Traffic 2 101
6.7 Prediction table reset facility 102
6.8 System-level design flow 104
6.9 PPE master and slave modules 106
6.10 Protocol-processing flow 106
6.11 Simulated systems 108
6.12 PPE processing model 109
6.13 PPE latency 111
6.14 Latency reduction 111
6.15 Individual processor workload 112
6.16 Aggregated processor workload 113
6.17 Arbitration- / bus- and memory-load 114
6.18 PPE throughput 115

7.1 Output data movement model 119
7.2 Priority + weighted fair queuing 120
7.3 Evaluation Methodology 121
7.4 Queuing delay EF queue 123
7.5 Delay jitter reduction 124
7.6 Network architecture 126
7.7 Delay portions 128

List of Tables

2.1 Protocol breakdown MCI Worldcom 1997 37

2.2 Protocol breakdown Sprintlab 2002 38

5.1 Most-frequently used entries 77

5.2 Protocol-stack identifier 80

6.1 Deterministic and randomized simulation traffic 94

6.2 Deterministic and randomized simulation traffic 95

6.3 Impact of initialization packets 99

6.4 Computation cycles and number of compare instructions for prediction computation 99

6.5 Randomized simulation traffic 103

6.6 Memory access times 107

6.7 Simulation traffic 110

6.8 Mean protocol-processing latency 110

6.9 NL-PU workload for UDP-IPv6-G-E 116

7.1 Mean delay and jitter 122

7.2 Processing delays 126

7.3 Number of complex processing router 127

7.4 Propagation delay 129

7.5 Delay values 129

Acronyms

APT	Additional Processing Time
ARP	Address Resolution Protocol
ARPA	Advanced Research Projects Agency
ATM	Asynchronous Transfer Mode
BE	Best- Effort
CMT	Coarse-grained Multi-Threading
CoS	Class-of-Service
CP	Control Point
CPG	Conditional Process Graph
DHT	Decode History Table
DSCP	DiffServ CodePoint
EDA	Electronic Design Automation
EF	Expedited Forwarding
FMT	Fine-grained Multi-Threading
G-E	Gigabit Ethernet
GEANT	Gigabit European Academic NeTwork
GPS	Generalized Processor Sharing
HDLC	High Level Data Link Control
ICMP	Internet Control Message Protocol
ICMPv6	Internet Control Message Protocol version 6
IETF	Internet Engineering Task Force
ILP	Integrated Layer Processing
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISO	International Organization for Standardization
ISP	Internet Service Provider
L2	Layer 2
L3	Layer 3
L4	Layer 4
MAC	Medium Access Control
MMM	Master-Master-Message

MP	Multi Processor
MP	Multiprocessing
MPLS	Multiprotocol Label Switching
MT	Multi-Threading
NAP	Network Access Points
NL-PU	Network Layer-Processing Unit
NP	Network Processor
OC	Optical Carrier
OSI	Open System Interconnection
OSPF	Open Short Path First
PCI	Protocol Control Information
PDU	Protocol Data Units
POP	Points-of-Presence
PPU	Prediction Processing Unit
PSSW	Protocol Stack Status Word
PU	Processing Unit
QoS	Quality-of-Service
RARP	Reverse Address Resolution Protocol
RTP	Real-time Transport Protocol
SEQUIN	SErvice QUality across Independently managed Networks
SLA	Service Level Agreement
SMT	Simultaneous Multi-Threading
SONET	Synchronous Optical NETwork
TCP	Transport Control Protocol
UDP	User Datagram Protocol
VMTP	Versatile Message Transaction Protocol
VPN	Virtual Private Network
WFQ	Weighted Fair Queuing
WRR	Weighted Round Robin
WWW	World Wide Web
XTP	Xpress Transfer Protocol

Chapter 1

Introduction

1.1 Internetworking

1.1.1 Network architectures

For more than three decades, data communication has been an essential component of computing. Driven by the US Defense Department Agency, a four node network was established in 1969 between UCLA, Stanford Research Institute, UC Santa Barbara and the University of Utah [1]. In the following years, linking autonomous local networks together became a main challenge. The intention was to share information of common interest. Unfortunately, difficulties arose due to heterogeneous physical networks with diverse hardware and software technologies. In addition, some applications created the demand for high-speed communication, which was not feasible by networks that span large distances [2].

A new technology, called *internetworking*, was created to satisfy these needs. It hides details of the underlying network hardware and allows interaction between different devices that comply with standardized internetworking specifications. The referring network, called *Internet*, links local networks together at interconnection nodes. For this purpose, network nodes include forwarding devices, i.e. routers. The Internet was developed on the principle of *packet-switched*, i.e. connectionless, datagram delivery. Each data unit, i.e. a datagram, is routed hop-by-hop towards its destination according to an inherent unique destination address which is assigned to each end device.

The access to the Internet backbone is depicted in Figure 1.1.

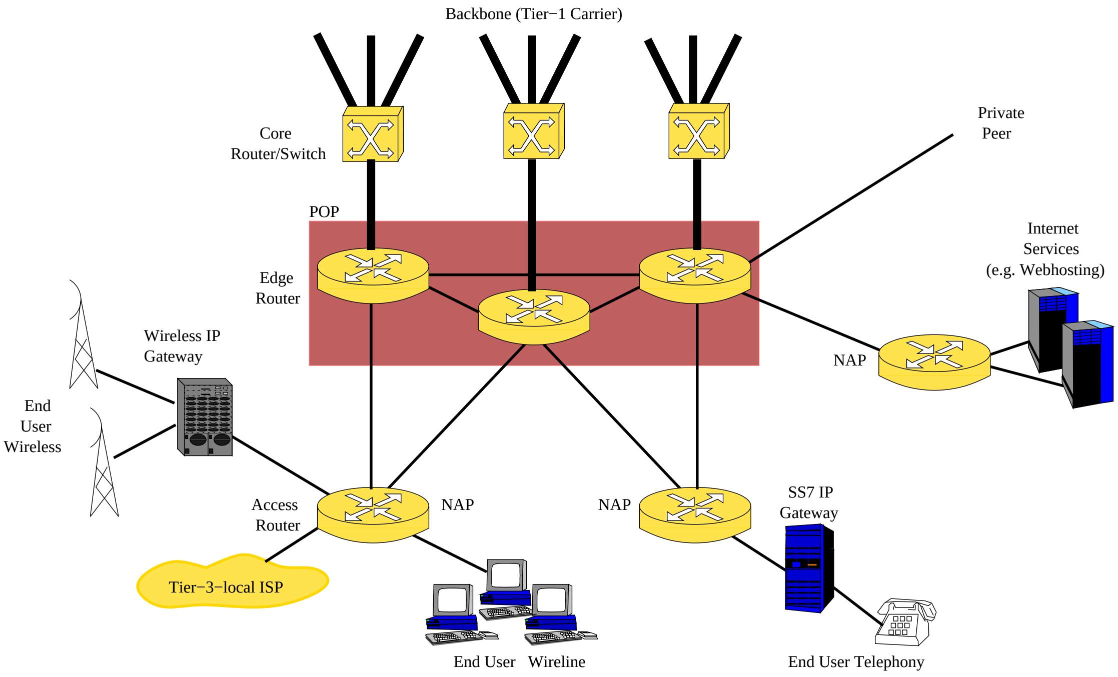


Figure 1.1: Internet Topology

Data and voice communication subscribers are connected either by wireline or wireless at Network Access Points (NAP). Furthermore, server farms can deliver content on request to end users. In order to access the backbone network on top, a set of nodes, known as Points-of-Presence (*POP*), are connected at the network edge. Communication over the Internet will be extended by voice applications, such as IP telephony. A POP has links to private peers, large company networks, regional tier-2 [1] Internet Service Provider (ISP) or local tier-3 ISPs. Tier-1 carrier represent the global backbone.

1.1.2 Communication reference model and service architectures models

The *Reference Model of Open System Interconnection* was defined to create a conceptual model for communication in an open communication infrastructure [3]. It is standardized as the ISO/OSI reference model of communication [4].

It comprises three abstraction levels [5]: An *architecture model*, which is built up of seven layers, describes the communication flow in an open system [6]. In order to partition the complete communication tasks, each layer provides well-defined *communication services*.

[1] tier: from medieval French *tire*: rank

In addition, each layer represents an autonomous unit. Communication between the layers is accomplished through well-defined calls and replies, so-called *protocols*. Figure 1.2 illustrates a communication architecture model with two end systems and a transmit system between.

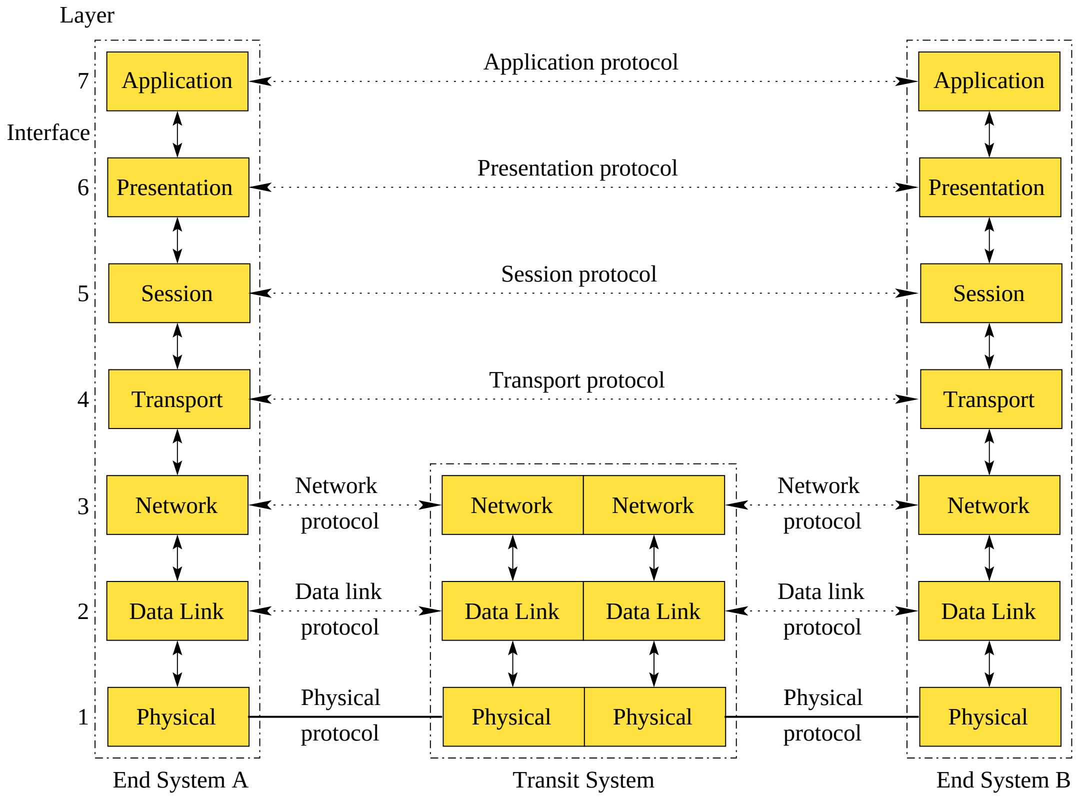


Figure 1.2: Architecture model according to ISO/OSI Reference Model

Each layer above layer three of end system *A* communicates with the same layer of end system *B*. Due to the lack of a horizontal link, the virtual connection between these two *peers* is enabled by vertical communication services. Each layer encapsulates user data into protocol control information (PCI) and forwards it top-down to the layer below. Transmission takes place over the physical medium. At the receiving side, i.e. end system *B*, user data is applied to the application layer bottom-up by decapsulation of the PCI.

The transmit system is intended for data forwarding between end systems, i.e. the source and destination system of the communication data. In order to check the transmission data and to determine the path towards the destination, layer one to three are involved.

The widely used protocol suite of the Internet is Transport Control Protocol (TCP) / Internet Protocol (IP). It was developed by the US Department of Defense as part of the ARPA (Advanced Research Projects Agency) technology [2]. IP implements the functionality of the network layer. TCP is one of two transport layer protocols in a TCP/IP network and provides a connection-oriented and reliable transport service. The other, the user datagram protocol (UDP) provides connectionless and unreliable transport service. For network control purposes, the protocol suite comprises protocols such as Address Resolution Protocol (ARP), Reverse Address Resolution Protocol (RARP), Internet Control Message Protocol (ICMP) and other.

Based on this model, the Internet offers flexibility and performance for current applications and services. The supply of audio and video communication as well as the support of reliable transactions, e.g. interactive games and stock trading, extends the traditional demand of *best-effort* transmission to the network infrastructure. These kinds of multimedia applications produce a great amount of burstiness within the network which makes it difficult to satisfy network subscribers requirements [7]. In addition to this, dynamic and efficient usage of network resources is an essential aspect. In order to provide end-to-end performance in terms of throughput, delay, jitter and congestion, a Quality of Service (QoS) architecture model was introduced.

Apart from the integrated services architecture model (IntServ) which enables per-flow QoS [8], two essential models for QoS provisioning are *Differentiated Services (DiffServ)* architecture and *Multiprotocol Label Switching (MPLS)*.

Differentiated Services

The DiffServ architecture was defined by the IETF [9] [10]. Its main intention is to overcome the scalability problem of the Integrated Service Architecture (Intserv) and to provide Class-of-Service (CoS) support [11].

Incoming traffic to a DiffServ network is classified, policed and possibly conditioned at the DiffServ ingress, i.e. the edge. Furthermore, the traffic is assigned to a behavior aggregate, which is identified by a DiffServ code-point (DSCP). Inside the DiffServ domain, traffic is forwarded according to the per-hop behavior (PHB) associated with the DSCP. Figure 1.3 illustrates the DiffServ packet forwarding model for packets with different behavior aggregates (premium, gold, silver, bronze).

Processing complexity in the DiffServ domain is reduced to increase packet throughput through fast but simple backbone routers. The underlying idea is the common design principle that protocol processing complexity is moved to the network border [12] where throughput requirements are moderate.

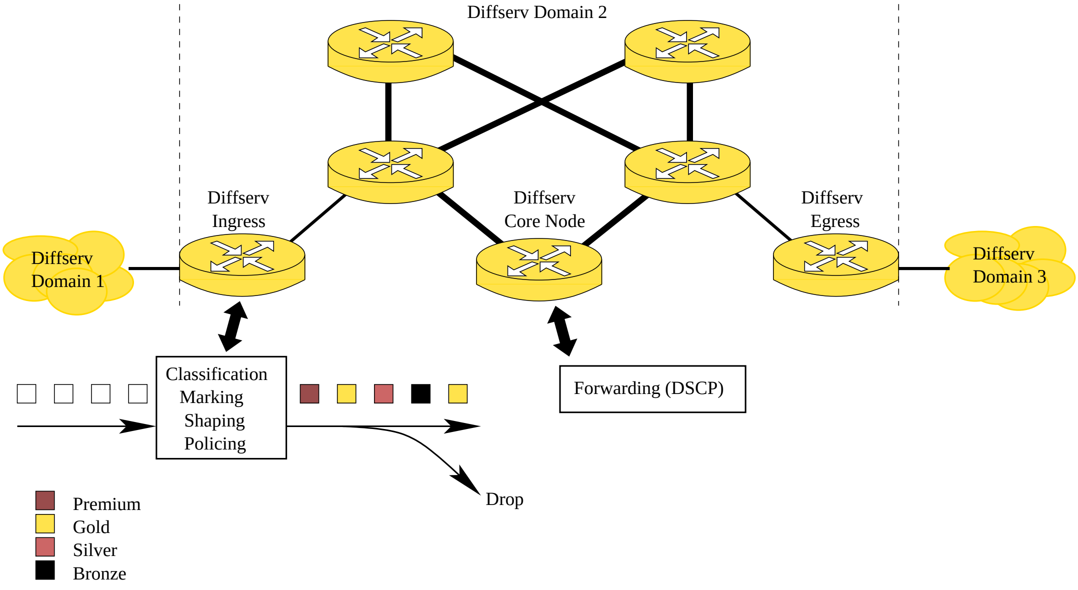


Figure 1.3: Differentiated Services Architecture

Multiprotocol Label Switching

Orthogonal to the node-level congestion avoidance mechanism of the DiffServ architecture, Multiprotocol Label Switching provides QoS through an integration of the label swapping forwarding paradigm with network layer routing. MPLS has been standardized by the IETF [13]. Figure 1.4 illustrates a MPLS domain.

The bindings between Forwarding Equivalence Classes (FEC) and labels are distributed among participating MPLS devices, i.e. label switch routers. Explicit paths, so-called *label switched path* (LSP) are determined and established subsequently. Packets which enter in am MPLS domain are labeled and assigned to a FEC. This corresponding label refers to the LSP. Packets with same FEQ are treated identically and consequently follow the same path through the domain. If a packet is forwarded to the next hop, the label is sent along with it. At subsequent hops, no complex packet header analysis has to be done. Instead, the current label is replaced by a new one which indicates the next hop. With regard to the OSI reference model mentioned above, MPLS is located between layer two and three. Thus, MPLS makes it possible to switch traffic through the network. At the MPLS egress node, the label is stripped off and the packet is forwarded to its final destination.

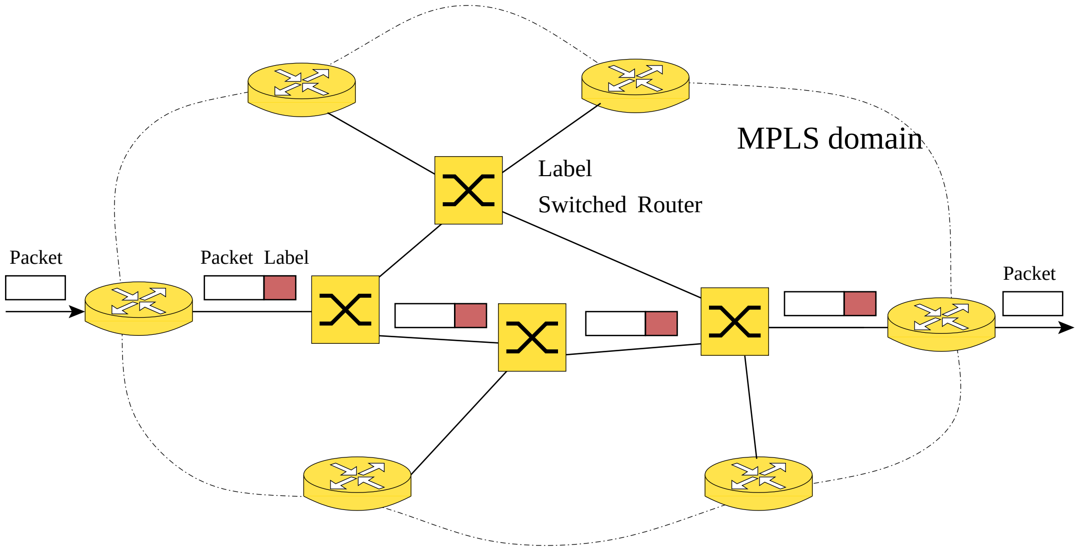


Figure 1.4: Multiprotocol Label Switching

Virtual Private Network

In order to provide secure data transport for authorized user groups in a public network, virtual private networks (VPN) have been standardized [14]. An IPSec[2] VPN is depicted in Figure 1.5.

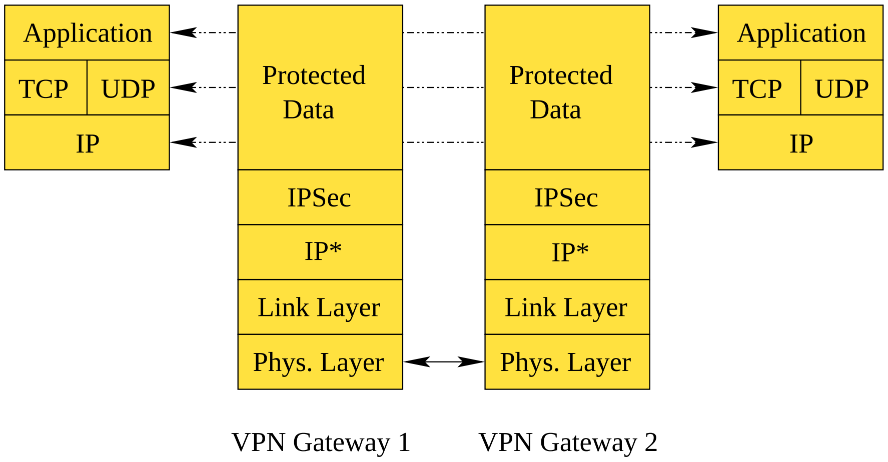


Figure 1.5: Virtual Private Network with IPSec

IPSec VPNs, also called Layer-3 VPNs, allow secure transmission in an IP network through

[2] IP-Security

data encryption. Beside several encryption standards, two encryption modes are supported: transport and tunnel mode. While the former encrypts only the payload of a packet, the latter encrypts both the header and the payload. In tunnel mode, which can be used in conjunction with Layer-2 Tunneling Protocol (L2TP), a new IP header IP^* is added with a VPN source address. Finally after transmission, decryption is done in the gateway at receiver side. Other types of VPNs are Layer-2 VPN and MPLS VPNs, respectively.

1.1.3 Network performance metrics

Key performance metrics of packet-switched networks are *latency*, *delay jitter*, *throughput*, *cost*, *loss* and *reliability* [15] [16]. The end-to-end latency of a packet-switched network, is illustrated in Figure 4.1.

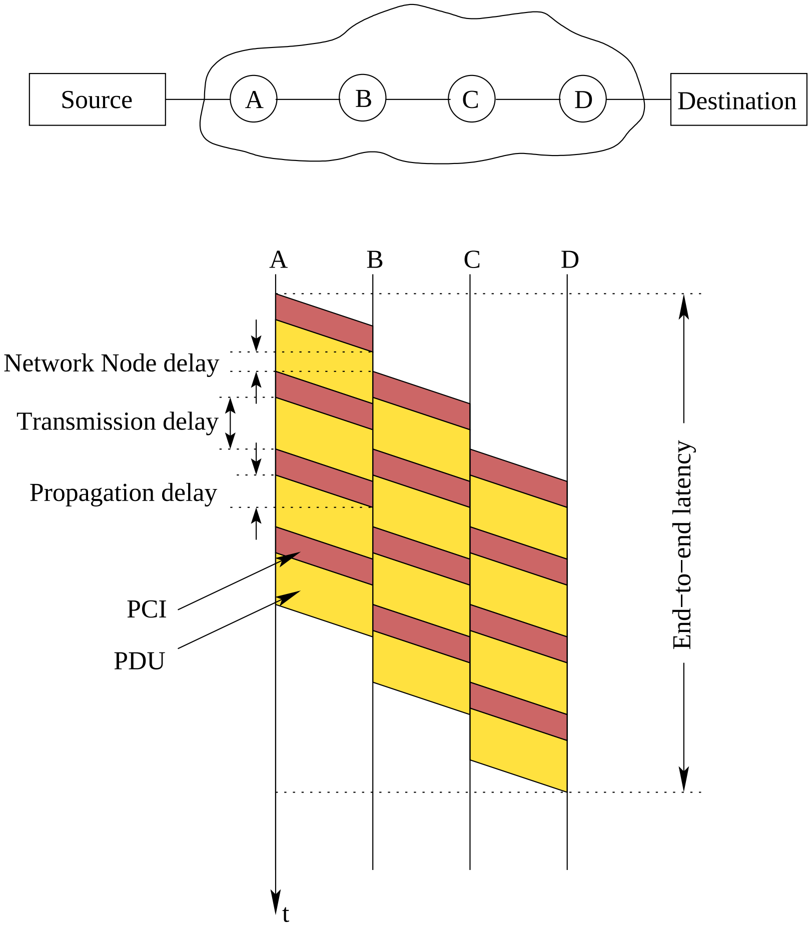


Figure 1.6: End-to-end-Latency

In the figure, data is decomposed, i.e. fragmentated, into four protocol data units (PDU) and a protocol control information (PCI) is assigned to each before the transmission. The transmission latency between data source and destination is composed of [17]:

- propagation delays of the transmission medium
- transmission delays which depend on data sizes and data rates
- network node delays which result from packet processing delays and internal queuing delays

1.2 Network processors

The first generation of routers in the 1980s, so-called store-and-forward routers, commonly used a bus-based computer system with a single central processing unit (CPU), a main memory and multiple network interfaces [18]. On the CPU, an embedded operating system was running, which was especially designed for routing purposes. Before forwarding, each packet is stored in the memory. An essential drawback of it is a performance limitation which results from a software implementation and the memory bandwidth.

In order to track with increasing transmission rates, several modifications concerning the architecture have been done in the meantime. The number of CPUs was extended from one to multiple to overcome the limited packet throughput. Route caches have been integrated in the system to provide frequently-used routing information quickly. The workload of the shared bus has been reduced by a distribution of CPUs, route caches and memories onto separate network interface cards. However, it still represents a bottleneck. Switch-based router architectures replaced the shared bus by a switch device, which enables fast packet transfer from the incoming port to the destination port.

During the last years, the demand for higher throughput up to 10 Gbit/s and more, flexibility requirements with regard to supported protocols and services, as well as economic imperatives of shortened hardware and software development cycles enforced alternative router designs with *network processors* (NP) as packet forwarding devices. In 1.7, a typical current router architecture is depicted [19].

On a backplane, multiple line cards are plugged in. External connection and framing functionality is provided by line interfaces. NPs accomplish packet analysis, routing lookups, dedicated packet handling, e.g. support of differentiated services, and queuing. Packet buffering and transfer to a switch fabric are done likewise. A system processor is used for control-point functions, e.g. routing table updates.

Even though the expression “network processor” is not standardized, common properties of these devices normally are [20]:

- An NP is a programmable device that is optimized for packet processing.

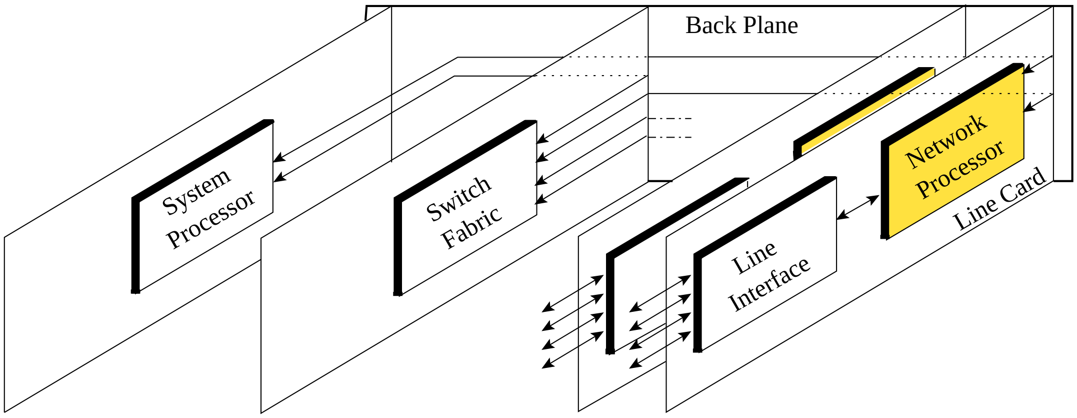


Figure 1.7: Router architecture

- An NP is designed for entire protocol-processing from layer 2/3 up to layer 7.
- An NP executes networking tasks at high speed.
- Individual packets can be processed fairly independently.
- An NP can possess embedded coprocessor blocks for accelerated processing of dedicated functionalities.

Figure 1.8 illustrates main building blocks of a typical current NP [20].

Multiple independent packet engines are used for fast data plane processing. This comprises header parsing, header analysis and modification as well as classification and routing. Packets are received and transmitted to the line interface by media access control (MAC) devices that can be implemented on-chip. Fast access to routing and classification information is provided by a search engine. External and internal memory devices are typically in the dimension of several megabytes. The corresponding device interfaces have to offer high bandwidth. A fabric interface either connects NPs with an external switch fabric or two NPs directly. Currently, packet memories range in sizes up to 512 megabytes. They are used to hold packet headers and payloads. Memories furthermore enable multiple transmission queues according to different traffic priority levels. A control processor which can be either on-chip or external, executes control plane tasks, e.g. the initialization of the NP or routing table updates.

Without being exhaustive, two examples for network processors are the Intel IXP 1200 and the IBM Power Network Processor 4G3. The IXP 1200 includes an on-chip array of six 32-bit micro-programmable RISC processor engines. Based on a synchronous multithreading architecture, each micro-engine supports 4 contexts [21]. Additionally, an integrated StrongArm processor core is responsible for control plane processing. A separate hash engine performs polynomial hash on up to three values at once with the intention of fast table

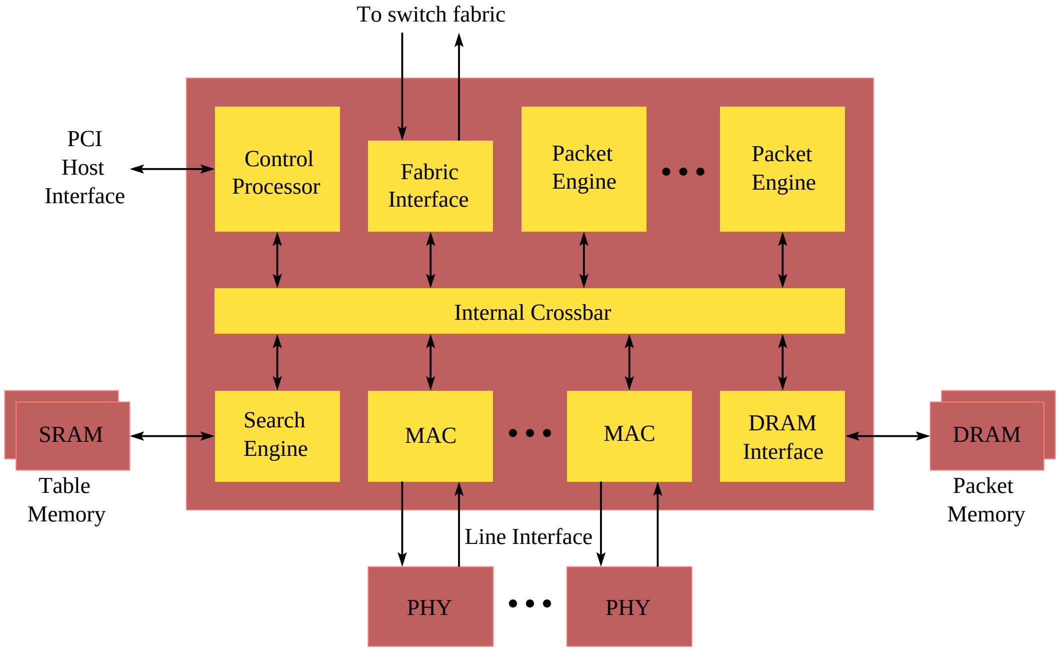


Figure 1.8: Network Processor Block Diagram

lookups. The chip possesses neither a direct switch fabric interface nor coprocessors for accelerated execution of functions such as classification and queuing. IBM NP combines a PowerPC CPU with 16 RISC packet engines [22]. As an essential extension in comparison to the IXP 1200, four table search engines and 56 coprocessor units are contained in the IBM NP. A switch fabric interface furthermore allows a direct connection from the NP to a switch fabric.

1.3 Motivation

The main objective of current network processors is to cope with the demand for high packet throughput [20]. In order to provide data rates up to OC-192, the fast processing path is realized by programmable processor cores which have an optimized architecture for packet processing. Multiprocessing and multithreading are two implementation techniques which can be commonly found in current NPs. Multiprocessing implementations realize packet processing in parallel execution units. As a result, the packet throughput can be increased. Multithreading is used in microprocessors to hide latency which results from access times of processor external devices. In NPs, multithreading processor cores perform program context switches during accesses to external packet and routing information memories take place. Thus, no processor stall times appear.

However, the processing the networking functionalities, such as checksum calculation, IP next hop lookup etc., commonly follows a sequential flow [23]. The reason for this sequential processing model are data dependencies which result from the structure of a packet protocol control information. The type of the transport layer is derived from a header field of the network layer, for example. Consequently, protocol-processing of the transport layer cannot be started until the network layer has been processed and the protocol layer type for layer 4 has been derived. Thus, NPs which apply sequential processing exhibit longer processing delays for packet handling than parallel processing flows.

This thesis challenges if a parallel execution model that exploits data speculation can be efficiently applied to protocol-processing in network nodes of the Internet. The associated goal is to achieve significantly reduced processing delays between the incoming and outgoing interface of a network processor compared to traditional sequential NP implementations. The proposed model derives basic principles from an efficient processing methodology in microprocessors, called branch prediction. The scheme is applied to reduce execution times of programs through a result prediction of single instructions and speculative processing of succeeding instructions [24]. However, it has to be noted that the intention of this work does not merely represent an adaptation of instruction prediction methods of the microprocessor domain to the application field of networking. Instead, it comprises a concept that performs a prediction on a higher level combined with the speculative execution of processing tasks.

1.4 Structure of this thesis

The structure of the dissertation is as follows:

Chapter 2 presents related work in protocol processing. At the beginning, the packet processing chain, which is composed of different tasks, will be introduced. In order to obtain an increased protocol-processing performance, different protocol specifications and protocol implementations were proposed. Their benefits as well as their weaknesses are observed. After this, an introduction into principles of data prediction is given and research concerning network traffic characteristics is presented.

As a main part of this thesis, **chapter 3** introduces a new protocol processing concept for network node devices. It consists of two parts, namely a *protocol stack prediction*, which predicts future protocol traffic based on an a-priory knowledge, and *speculative data processing*, which speculatively executes tasks of the protocol processing chain. The associated benefit is a less mean processing delay.

An abstract verification is done in **chapter 4** to work out benefits of the proposed concept. At this step, details about the implementation as well as architectural considerations are out of scope. In order to obtain general insights concerning the latency as well as processing resource costs, networking specific restrictions, e.g. characteristics of particular protocol types, are ignored. Essential parameters are extracted and their influence on the concept is separately evaluated in individual studies.

In order to evaluate the concept under consideration of internetworking-specific constraints, a system architecture is defined in **chapter 5**. The architecture implements the functionality of an network processor which is applied in an Internet edge router. The functionality of the data-plane, which contains the high-speed protocol processing path, and the prediction unit are specified and their architectural implementation is presented. It should be emphasized, that the predefined architecture neither lay claim to be optimized nor that it represents a competitive solution to disposable network processors. Instead, it represents an exemplary architecture, which can be used for a performance evaluation. It helps to find out potential limitations and performance bottlenecks of the speculative concept.

Chapter 6 describes the applied test methodology for the speculative system architecture of chapter 5 and discusses the obtained simulation results of the packet-processing element and the prediction unit.

Chapter 7 considers the application of the proposed processing concept in networking nodes and its impacts on end-to-end communication. In order to compensate the delay jitter, which results from occurring mispredictions, a well-suited module for output scheduling is proposed and evaluated.

Chapter 8 concludes the work with the scientific contribution and gives an outlook for further optimizations of the proposed concept.

Chapter 2

Protocol-processing methodologies

2.1 Introduction

For more than three decades, an extensive research has been done in protocol processing to cope with increasing throughput requirements, different applications which came along and changing network technologies [25]. The explosive growth in network bandwidth and services exceeds the performance increase of traditional microprocessors [26]. For data packets of 64 bytes and a bandwidth of OC-192, an NP has to perform packet-processing in only 52 ns. In case of OC-768 merely 13 ns are available. Additional services which have to be optionally supported, require a programmable or configurable NP architecture. These aspects rise a demand for alternative protocol-processing methodologies that represent more efficient concepts compared to general-purpose processor implementations. These concepts can be classified into *protocol specifications* and *protocol implementations* [27]. While the former tries to avoid the complexity of traditional network protocols, the latter has the goal to efficiently map a given protocol or a protocol suite on an underlying system architecture.

Some of the approaches, e.g. protocol specifications, are strictly dedicated to networking applications. With the introduction of NPs, embedded microprocessor cores were applied to packet-processing tasks. Thus, recent approaches commonly try to adapt efficient methodologies of microprocessors , e.g. multithreading and multiprocessing, for use in network processors to increase the performance.

After an introduction of the packet-processing chain and networking functionalities which have to be handled by a network processor, the state-of-the-art methodologies in protocol-processing will be shown by an overview of various protocol specifications and protocol implementations. The main intention of this chapter is to illustrate their capabilities and limitations.

Fundamentals of predictive processing models are shown afterwards. They are indispensable for an application of data prediction. Branch prediction, which is an ubiquitous method for acceleration of program execution in current microprocessors, will be explained.

Finally, a summary highlights drawbacks of different protocol-processing methods that are currently used with respect to short processing latencies.

2.2 Packet-processing

In network nodes, packet processing follows frame processing which is performed by a framer or a Medium Access Controller (MAC) . Thus, considering the OSI reference model, packet-processing comprises parts of protocol layer two, layer three and higher. The main task is to forward packets in direction towards their destination. It is based on one of two transmission paradigms: *connection-oriented (end-to-end)* or *connectionless (hop-by-hop)*. The end-to-end model arranges the route from data source through the network to the sink before transmissions of user data take place. Contrary to this, hop-by-hop determines the route to the destination according to routing protocol information in each network node. The decision is made there over which connected interface of the device the data has to be sent.

As depicted in Figure 2.1, packet processing is divided into *control-plane* and *data-plane* processing.

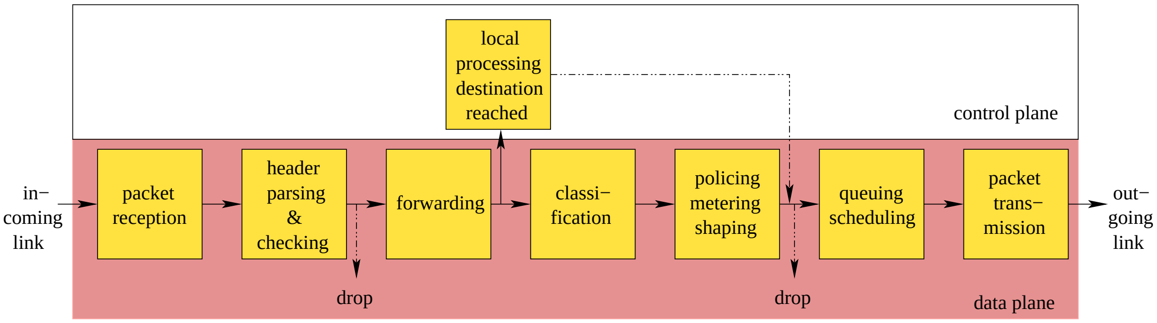


Figure 2.1: Packet Processing Chain

Both cover different protocol-processing tasks. The control-plane performs network administration tasks, e.g. routing information exchange and network administration.

Data-plane processing refers to data transfer towards the destination of a user packet. The packet is sent from the incoming interface, along the data-plane processing path to the outgoing interface. The target of the data is indicated by the destination address which is part of the protocol control information (PCI).

Processing tasks contain computations, e.g. algebraic calculations, which require input values. These values could be a part of the PCI of a packet. Otherwise, information, e.g. network node states and connection parameters, is kept in the network node itself. The output of processing tasks delivers information concerning the further handling of the packet, e.g. whether the packet has to be forwarded or dropped. This dependency of required input data consequently forces a sequential flow of execution.

In order to achieve required network capabilities in terms of performance and transfer quality, a common paradigm is to keep the network core simple but fast. Complex packet-processing tasks are shifted to the network edge [28]. Thus, the packet-processing chain within the network core possesses a simplified and reduced complexity. The compliance of a service level agreement (SLA) between a customer and a service provider is verified at the network domain ingress. Time-consuming functions, e.g. a multi-field classification of the packet, are processed there. The classification which is subsequently performed in the core, merely considers the type-of-service field of the PCI.

Subsequently, main processing tasks of an edge-router, as shown in Figure 2.1, will be explained in detail. Considering performance requirements of a packet processor device, only tasks of the data-plane will be treated. The following description is based on the TCP/IP protocol suite.

2.2.1 Packet reception

A link-layer device, e.g. a framer or MAC, transfers received packets to the packet processor. The packet reception unit of the NP is responsible for tasks such as packet reassembly, identification of the physical input port and memory allocation for packet buffering [29].

2.2.2 Header parsing & checking

In order to decide how to handle an incoming packet, its PCI has to be gathered. According to the applied TCP/IP suite, the PCI consists of different protocol layer headers. The network layer protocol for the Internet is the Internet protocol. Transport layer protocol types of the TCP/IP suite are the transport control protocol and the user datagram protocol. In Figure 2.2, an IP packet is shown, which contains UDP as the encapsulated transport layer protocol.

Depending on the protocol-stack, each layer is built up of multiple fields which include information, e.g. source address, destination address, checksum, protocol identifiers. Header parsing extracts particular fields which can be found at predefined offsets from the header start address. However, optional fields cause a variable header length and, thus, increase the field localization effort. Checking is used for an evaluation of the packet header, i.e. whether packet data is corrupted or not.

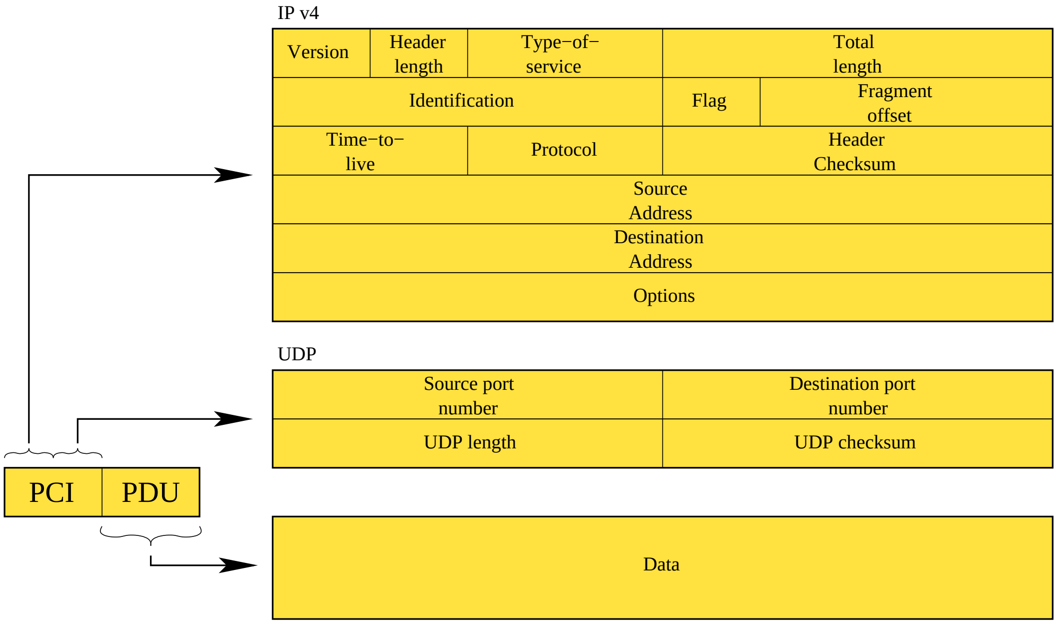


Figure 2.2: IPv4 packet

2.2.3 Forwarding

Forwarding is responsible for choosing the outgoing link for a packet in the direction to its destination. The execution of mandatory tasks, which are specified in [30], are part of the forwarding block as well. The routing decision is made based on the address entries of a routing table to which the forwarding unit has access. Routes are defined by an address of the destination network. This address is specified by an address prefix and a prefix length [31]. The next hop, to which IP packets are forwarded, is derived from the longest prefix match among available routes of the routing table. If the destination address corresponds to the network node address, the packet destination is reached. In this case, the transmission is terminated and local processing follows, e.g. control-protocol processing in the control-plane.

2.2.4 Classification

The determination to which traffic class an incoming packet belongs to, is done by classification. One or several fields of the PCI are extracted as input for the classification. Header fields that are commonly used in case of a multi-field classification at the network edge are source and destination address, physical input port, protocol types of the application layer, and the Type-of-service field [32]. The classifier returns an output value that indirectly characterizes the further treatment of the packet.

2.2.5 Policing, metering, shaping

Before operation, network customer and provider sign a Service Level Agreement (SLA) that specifies service guarantees. During operation, the compliance of the SLA has to be checked. The profile meter measures the actual flow traffic and determines for each packet whether it complies with the agreed traffic profile. In case of conformity with the SLA, the packet is handled without restrictions. Otherwise, the policer marks the packed as non-conforming. As a consequence, it can be degraded to a lower service level or even dropped.

In order to avoid violations of the SLA, incoming traffic can be shaped through an insertion of a packet delay instead of being dropped.

2.2.6 Queuing, scheduling, congestion avoidance

Packets that are admitted to transmission have to be gathered until the outgoing link scheduler activates transfer. Packets are buffered in one or multiple queues. In case of a single queue, packets are enqueued and dequeued in order of their arrival. However, additional per flow states of packets are not kept. Systems with multiple queues allow to handle separate traffic flows individually. Their intention in case of CoS support is a preferential treatment to preselected flows for differentiated service queues.

The number of queued packets has to be monitored due to a limited amount of buffer and memory space. In order to avoid packet congestion in a network processor, a queue manager is responsible for controlling queue states and for preventive packet discard in case of congestion.

The packet order and consequently the decision which of the buffered packets has to be transferred next to an outgoing link is made by a scheduler. The selection of the applied scheduling algorithm is based on criteria such as fairness, implementation complexity and quality-of-service guarantees.

The highest fairness is provided by Generalized Processor Sharing (GPS), which is an idealized fluid queuing model that ensures a bandwidth allocation among all backlogged sessions in a system [33] [34]. However, due to a finite granularity of queuing server resources and packet sizes, GPS is un-implementable [35]. Weighted round robin (WRR) serves a packet from each non-empty queue in turn. It can show unfairness for long periods of time and need to know mean packet sizes in advance [35]. Weighted fair queuing (WFQ) is a packet approximation algorithm of Generalized Processor Sharing (GPS) [33] [34]. WFQ serves packets in an increasing order of their virtual finish times in the analog GPS model. An essential benefit is that it is capable to handle variable packets sizes and queue weights. Queue weights are defined to assign a dedicated server priority to queues.

Some scheduling algorithms that are currently used in routers to provide CoS support are weighted-round robin, weighted fair queuing (WFQ) as well as derivatives of WFQ, e.g. class-based weighted fair queuing (CBWFQ) [29] [36].

2.2.7 Packet transmission

At the end of the processing chain, the packet transmission module transfers a packet to the succeeding device. This might be either a switch fabric or a link-layer device, e.g. a MAC. According to the underlying frame format which depends on the applied transmission media and the corresponding maximum transfer unit, packets might be divided into several fragments. The corresponding process of dividing a packet into several parts is called fragmentation.

2.2.8 Additional and optional tasks

Packet-processing devices may provide additional tasks which are not depicted in the Figure above. Some of them are listed below.

- *Load balancing* enables a distribution of packet transmission among different network routes.
- If the system is used as a filter, e.g. a firewall, *access control* prevents transmission of non-authorized traffic.
- *Data encryption* hides private data for non-authorized users. An example for network environments, where data encryption is applied, are virtual private networks.
- *Network address translation* converts private addresses into addresses of the public network.
- *Accounting* and *billing* enable monitoring and gathering of traffic statistic for charging customers and for network maintenance purposes.

2.3 Protocol specifications

2.3.1 Light-weight protocols

Commonly used network protocols have the objective to cover a wide variety of supported applications. For the transport layer, TCP was defined in 1974 to provide connection-oriented reliable delivery [37]. In 1980, UDP was added to enable connectionless transmission [38]. The underlying network layer protocol for both is IP. Contrary to these *general-purpose transport protocols* [25], light-weight protocols are designed to avoid the complexity of traditional protocols. In [39], protocol processing of layers above the media access layer is mentioned to be often the bottleneck in communication systems.

Light weight protocols, such as Xpress Transfer Protocol (XTP) and the Versatile Message Transaction Protocol (VMTP), were defined to meet performance requirements of low latency and delay restrictions for multi-media-applications and to overcome the sequential structure of standardized protocols [40].

Key issues in light-weight protocols are:

- Fixed header format to simplify header parsing
- Implicit connection setup to avoid a transmission of additional control packets that are required for explicit connection setup

Main differences of XTP compared to TCP are [41]:

- Orthogonal protocol functions with the intention to separate paradigms from policies: separation of communication paradigm (datagram, virtual circuit, transaction) from the error control policy employed
- Separation of rate control and flow control: Contrary to XTP, TCP does not provide a rate control, which is a congestion based concept in the network node. Both, TCP and XTP, however provide a flow control mechanism, which considers the buffer space from end-to-end perspective.
- Explicit multicast support

VMTP was designed as application-oriented transport protocol for message and object transfer and possesses multicast capability [42].

As a result of the evolution of transport protocols towards an application-orientation, light weight protocols have not gained broad relevance. Instead, commonly used protocols are TCP for reliable data transfer, UDP for unreliable transfer and the real-time transport protocol (RTP) for real-time communication [25].

2.3.2 Adaptive protocol-stacks

Fixed protocol-stacks may exhibit performance drawbacks due to the communication effort between protocol layers and duplicated protocol tasks across layers. Adaptive protocol stacks can be configured by a generation of a light-weight protocol stack implementation that correlates exactly with predefined application requirements [43]. In order to achieve an optimum adaptation, protocol requirements are specified in a formal language [44]. Instead of an approach that is focused on layer-orientation, a function-based communication model is proposed that considers the application [45] [46]. Redundant layer processing tasks are removed through a division of protocol-stacks into functions instead of layers.

The heterogeneous structure of the Internet as well as the multitude of distributed applications however prevent the generation of an ideal protocol-stack. Thus, adaptive protocol-stacks cannot be applied to achieve a significant processing delay reduction in current IP networks.

2.4 Protocol implementations

Early packet processing systems were commonly build of a bus-based general-purpose computer system with multiple network interfaces [18]. Each packet which has to be forwarded is stored in a main memory. With the ongoing evolution of the Internet in terms of throughput and supported applications, the requirements for efficient protocol-processing in network nodes rose significantly. Studies concerning performance bottlenecks of packet processing devices led to a classification of functions into *compute-bound* and *memory-bound*. While compute-bound tasks use processor resources exclusively, memory-bound tasks use processor resources combined with accesses to shared memory resources. The more memory-bound a transfer is, the more likely it is that memory contention between processors in a multi-processor system will limit speedup [47]. Essential time-consuming factors of protocol processing functions, e.g. the checksum and next hop calculation, are read and write access times of the memory. In [48], it is concluded that the performance of protocol processing devices will most likely be limited by the capacity of shared busses and the memory system. Furthermore, problems which result from memory-bound functions are intensified by the fact that access latencies of memory systems using DRAMs decrease at a slower speed than CPU clock rates increase [49].

In order to cope with the performance requirements and under consideration of the aspects mentioned above, current protocol implementations are presented below.

2.4.1 Integrated-layer processing

According to [51], data manipulation is one of the costliest aspects of data transmission due to packet load and store operations of RISC system architectures. Well-known examples are encryption, checksum computation or compression. Integrated Layer Processing (ILP) has been introduced to reduce the number of manipulations [52]. Traditional implementations pass packets from one protocol layer to another through exploitation of a system data structure with several buffers as depicted in Figure 2.3 (a).

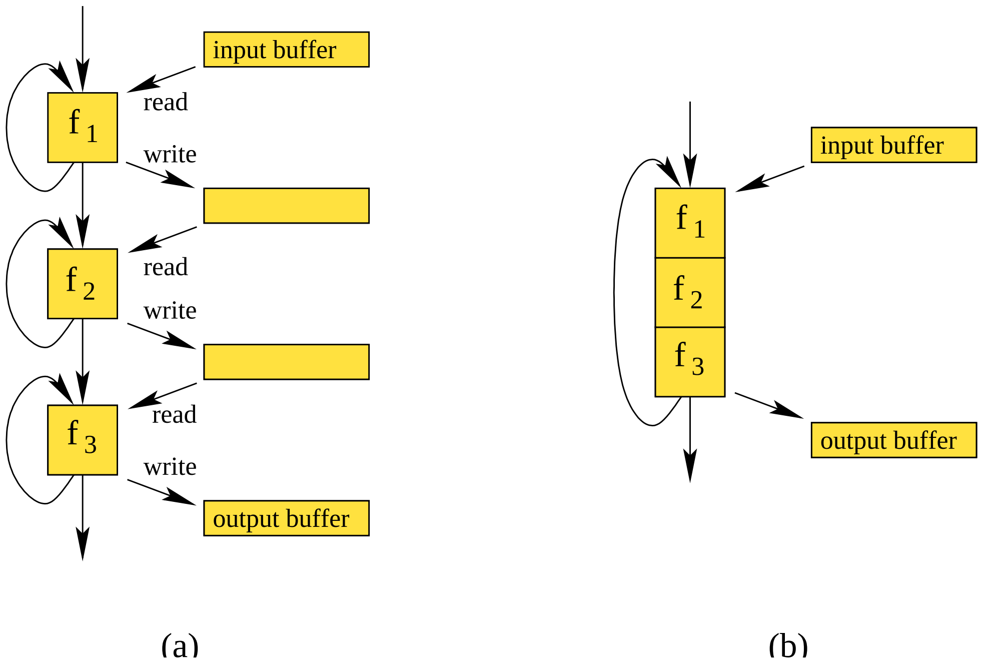


Figure 2.3: Integrated-layer Processing

By contrast, ILP integrates a series of protocols and the corresponding functions f_i in order to combine accesses to packet data as shown in Figure 2.3 (b) [53]. The intention of the concept is to achieve a higher data throughput.

In order to evaluate ILP, [54] points out that the concept has a limited applicability to complex functions which are used for data manipulations. Considering the protocol layer hierarchy, i.e. the encapsulation of higher layers into lower ones, only a few tasks of the packet which are covered by all manipulation functions can be processed in an integrated implementation. As a consequence, no significant reduction of the processing latency can be achieved.

2.4.2 Connection-oriented protocol-processing optimization

TCP has been introduced to provide a connection-oriented and reliable data transmission [55]. It comprises a TCP connection establishment and termination as well as an interactive or bulk data flow. One evidence for the reliability is the generation of an acknowledgment packet by a receiving host at one end of the TCP connection.

Optimization approaches which target on a performance increase of protocol-processing in end systems, try to exploit this interaction between the hosts at the ends. Statistically favored transmission states are factored out and will be handled as special cases. By a dedicated implementation of them, an accelerated protocol-processing is possible. The main two concepts are explained below.

Jacobsons' header prediction

The header prediction scheme which is proposed by Jacobson calculates a predicted header from a previous packet, which is assumed to be received next [56] [57]. It comprises a prediction statement about the next expected segment for a current connection which is the next in-sequence TCP data segment. This method is applied at the receiver site of data transmission and is used for connection-oriented traffic [55]. Occurring events at receiver site and the prediction flow are illustrated in Figure 2.4.

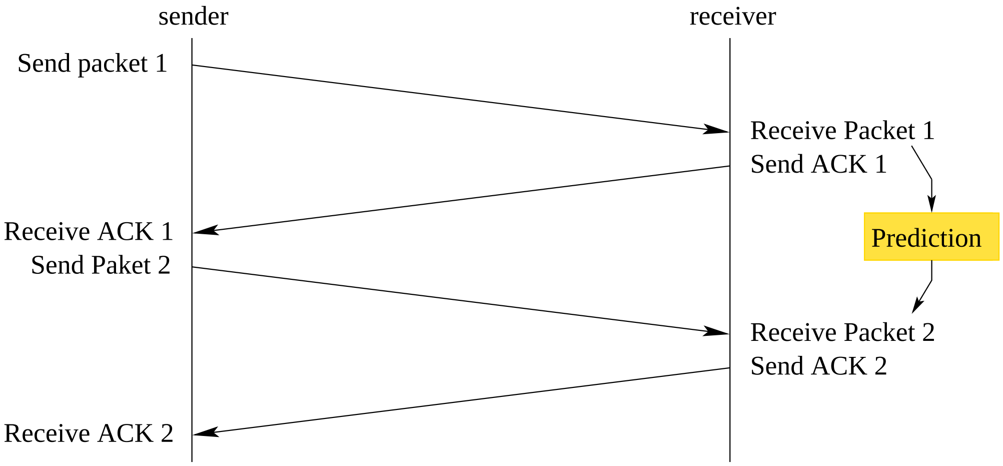


Figure 2.4: Jacobsons' header prediction

Woodsides' protocol bypass

In [58], a generalization of Jacobson's Header Prediction is given by Woodside. It extends the application of the prediction scheme to the data sender, i.e. if TCP is sending data, the next expected segment for this connection is an ACK for outstanding data.

Based on the 80 percent/20 percent-principle [59], packets which refer to frequent transmission states, bypass the common protocol-stack. Instead processing is done by a dedicated fast path. This path could be used for example for packets with identical PCI characteristics. In Figure 2.6, the implementation of the concept within an end system is shown.

The main operation that is performed by the concept is the increase of PDU numbers by one for each layer at sender and receiver side. In addition, an acknowledgment is sent after a receive event. Two key operations decide whether the bypass is taken or not: packet identification and bypass header match test. The second item ensures that data is for the

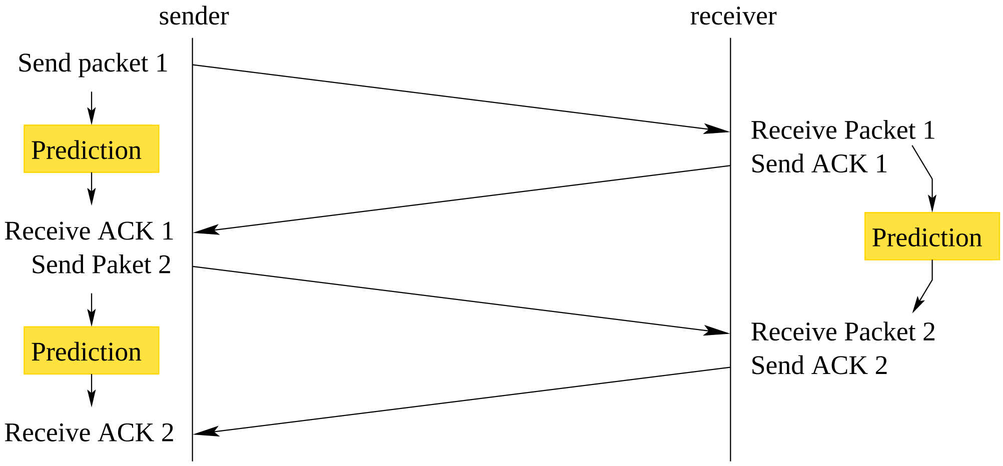


Figure 2.5: Woodsides' Protocol Bypassing Scheme

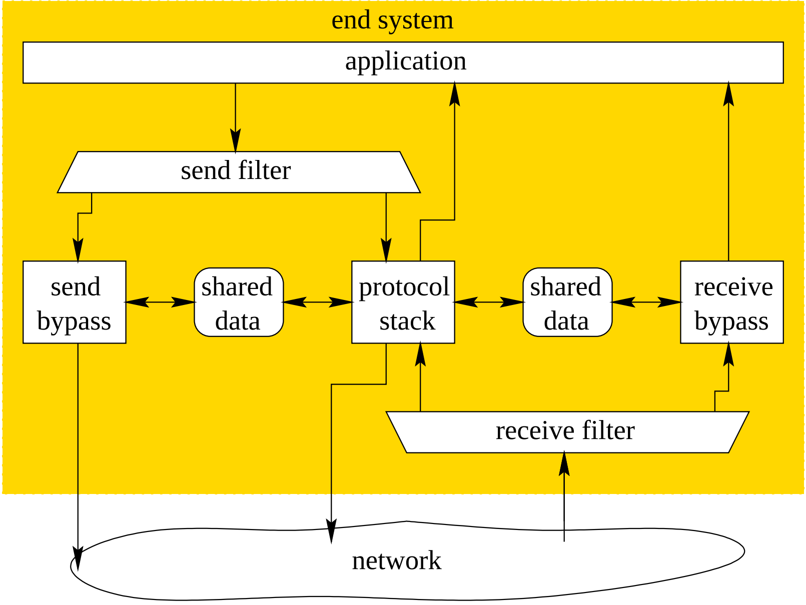


Figure 2.6: Protocol Bypass Architecture

right connection and provides the expected sequence number. The benefit of this approach is a significant increase of the system throughput and a reduction of processing delay at termination nodes.

Both approaches shown above allow a delay reduction for packets which bypass slow-path processing. Thus, for a period of time, a lower mean processing latency can be achieved. However, conceptual restrictions of the protocol bypass and TCP header prediction method with respect to an application in network nodes are:

1. The application of both concepts is limited to network end systems. Transit systems within the network are not supported [60].
2. The performed prediction is based on traffic connection states. Stateless transport protocols, e.g. UDP, are not supported. Furthermore, only packets which have a relation to the previous can be predicted.
3. In case of partial misprediction of packet data, no additional prediction is requested for remaining processing. This singularity of a bypass test operation limits the latency reduction.

2.4.3 Multiprocessing / Parallel protocol-processing

In order to cope with an increased effort for protocol-processing, Multiprocessing (MP) was introduced [61]. MP realizes the simultaneous processing of packets through multiple processing units that work in parallel.

A classification of parallel-processing architecture models that utilize MP can be found in [62]. Three different categories are defined that differ in the implemented parallelism. The kind of parallelism is expressed by the type of the replicated unit, e.g. per-protocol, per-connection, per-packet, per-layer and per-protocol function. The corresponding process architectures are categorized in horizontal, vertical and hybrid [63]. Although each of these process architectures has different structural characteristics, it is generally possible to implement the same protocol functionality.

Horizontal process architectures assign processing units (PU) to protocol layers or to protocol tasks. Each PU performs the appropriate protocol-processing operations on the data packet. Two kinds of horizontal process architectures are layer parallelism and functional parallelism as shown in Figure 2.7.

Layer parallelism is shown in Figure 2.7 (a). Each protocol layer is is assigned to a PU. Note that PUs are illustrated as dark rectangles in the following figure. Packets are transferred from one PU to another. According to [63], its main drawback is a fixed amount of parallelism. In Figure 2.7 (b), functional (task) parallelism is illustrated. Different protocol-processing tasks are performed in parallel by multiple PUs. While the use of multiple processing units potentially increase the performance, disadvantages are a synchronization overhead between the PUs.

Vertical process architectures assign processes of the operation system to connections or messages. Figure 2.8 shows two examples for vertical process architectures are connectional parallelism and message parallelism.

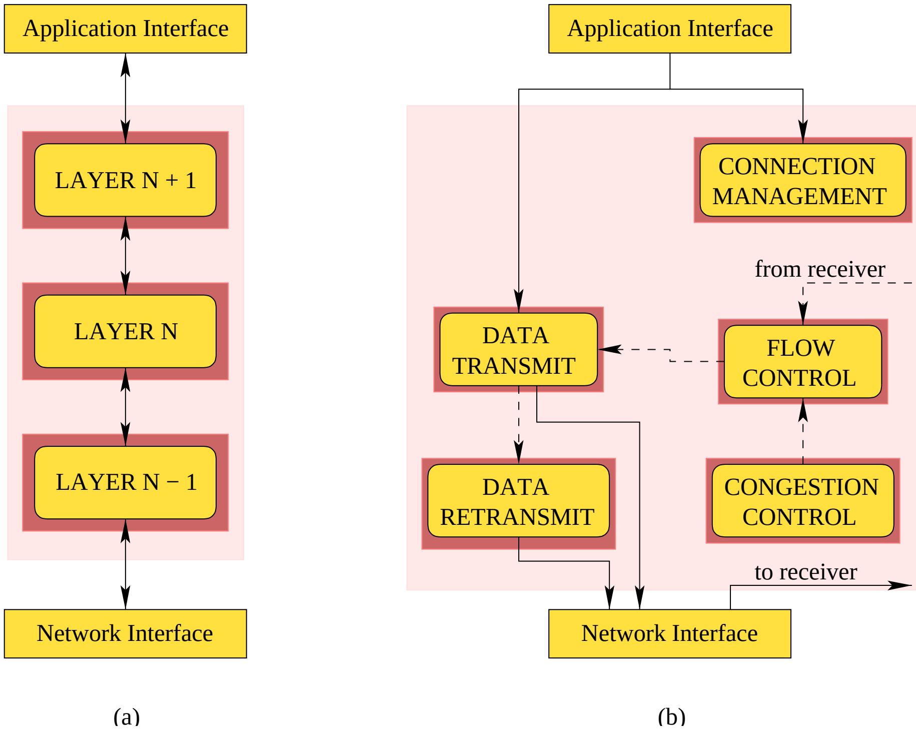


Figure 2.7: Horizontal Process Architectures

The connectional parallelism in Figure 2.8 (a) shows that different connections C_i are assigned to separate PUs. Each PU process all packets which refer to this connection. Connectional parallelism can be applied for example to network servers that offer connection-oriented services. However, connectionless traffic cannot not be supported by the concept. Message parallelism, which is shown in Figure 2.8 (b), associates a separate processing unit with one message, i.e. packet. The overhead that results from resource management and scheduling represents a main disadvantage of message parallelism [63].

Hybrid models realize horizontal and vertical parts together in one system. For instance, current network processors possess several programmable processing units which fulfill the majority of protocol processing tasks in message parallelism. In addition, hardwired coprocessor modules for accelerated execution of dedicated processing tasks, i.e. checksum calculation, correspond to functional parallelism.

As mentioned in the previous chapter, network processors and their architectures target on a throughput increase through the efficient use of system resources. They commonly consist of multiple processor cores that are used for packet-processing tasks. For NPs, two common models for multiprocessing have been established [64]. They differ in the deployment of the processor cores in the NP with regard to packet-processing tasks. The *run-to-completion* model in Figure 2.9 is characterized by the assignment of a packet to a

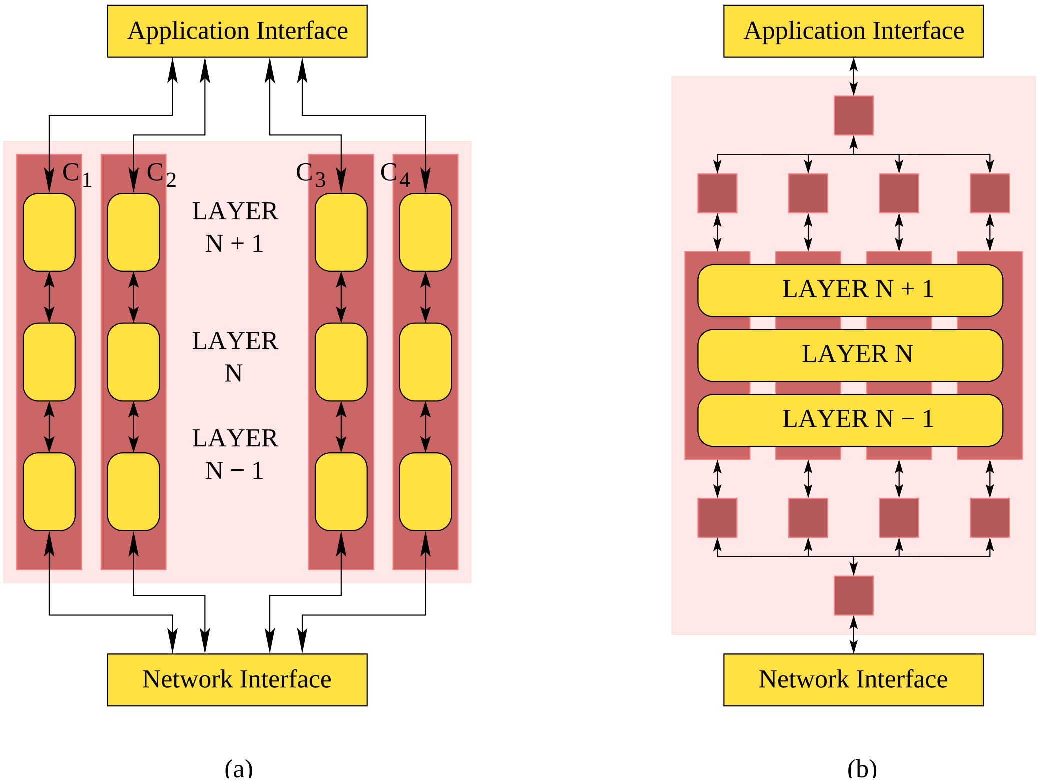


Figure 2.8: Vertical Process Architectures

single processor core. The complete packet processing is performed by this core.

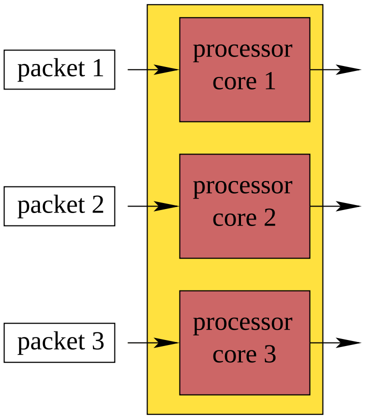


Figure 2.9: Run-to-completion model

The processing latency of a single packet depends on the processing model of the assigned processor core and on the interaction between cores in terms of accesses to shared resources.

The *systolic* model, which is a generalized form of *pipelining* [65], transfers a packet from one processor core to the next until processing is complete. As a common implementation component of CPUs, pipelining overlaps multiple processing steps in parallel [66]. These steps, which are called pipeline stages, are connected to form a pipe. Data units, which have to be executed, enter the pipe at one end, pass through and leave after completion of processing. For networking applications, pipelining can be applied on packet forwarding in two dimensions. Besides instruction pipelining, the protocol processing chain can be divided into individual functional stages, e.g. receive, classification, forwarding and transmit stage. During processing, a packet is assigned to one of these functional stages and passes through the functional pipeline. Figure 2.10 depicts a systolic model built up of three stages.

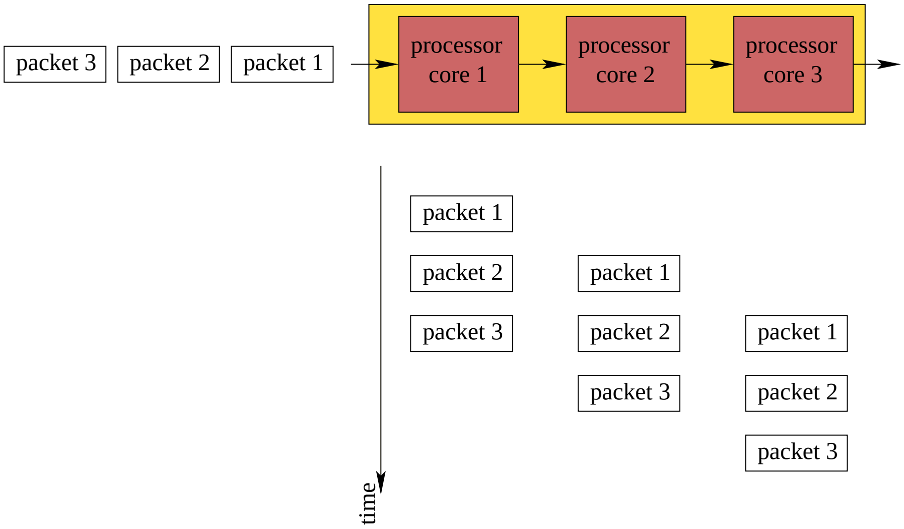


Figure 2.10: Systolic model

The main goal of the model is to achieve a throughput increase. However, the processing delay for the systolic processing model is given by the sum of the latencies for all pipeline stages and remains unchanged [67].

2.4.4 Multithreading

Multithreading (MT) was introduced in the area of microprocessors to increase the number of instructions per cycle. Its intention is to “hide” processing latency that results from operations which access devices external to the processor, e.g. an off-chip address lookup. It is enabled through the capability of context switches between threads to obtain an efficient load of CPU resources.

A *thread* is defined as a single sequential flow of control within a program [68] [69]. Each thread acts like a sequential program. A thread has a beginning, a sequence and an end. At any given time during the runtime of the thread, there is additionally a single point of execution. An n-way MT processor can execute as many different threads as it has support for, i.e. n, and possesses the capability to store the states of n threads. Multithreading can be divided into three categories:

- Coarse-grained multithreading (CMT)
- Fine-grained multithreading (FMT)
- Simultaneous multithreading (SMT)

Figure 2.11 illustrates CMT (a) and FMT (b) [70].

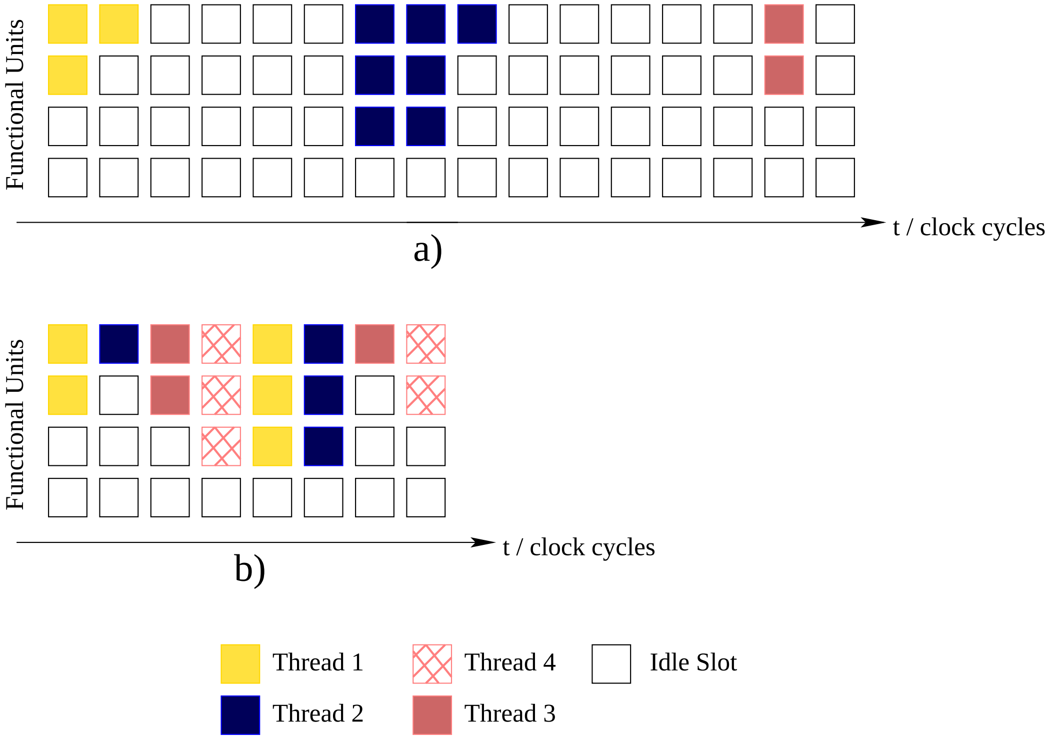


Figure 2.11: Coarse-grained and Fine-grained Multithreading

Both approaches execute one software thread at a time. When a switch to another thread is invoked, the processor saves the state of that thread and loads the state of the next thread.

A CMT architecture merely executes a single thread at a time, but does not possess on-chip register states for multiple threads. A switch between contexts requires a certain number time of processor cycles. For example, the multithreaded PowerPC processor described in [71] performs a context switch in 4 to 6 processor cycles. FMT enables the processor to switch between contexts on every cycle [72]. Thus, processor resources are shared at a finer granularity. Finally, SMT is a technique which combines the capabilities of fine-grained multithreading with the multiple-issue per instruction feature of super-scalar microprocessors [73] [24].

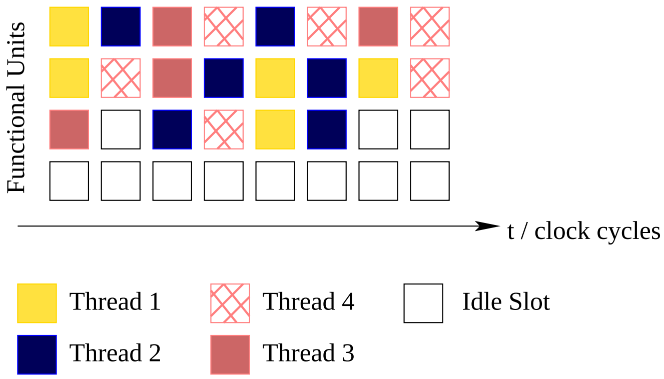


Figure 2.12: Simultaneous Multithreading

The majority of network processors uses multithreading (MT) as processing model for protocol-processing. Examples are given in [21] [22].

MT requires a considerable amount of hardware resources and complexity [74]. The essential advantage is an increased system performance for memory bound applications when multiple context threads share resources.

In order to analyze the processing time of a network processor that exploits MT, the MT processor system in Figure 2.13 is defined.

A packet, which is received by the system, is assigned to one of the two contexts in the PU. Figure 2.14 shows the processing flow for two packets.

For demonstration purposes, the complete protocol processing chain of a packet i is defined to three tasks Ti_j with $j \in \{a, b, c\}$. Tasks of one packet refer to the same thread. Each of the first two tasks initiate an access request to the shared memory (MEM). After the access is given by the bus arbiter, data are transferred from the PU through the bus to the memory (MEM) or in reverse direction.

It is furthermore assumed that data dependencies in the packets exist, which concern their protocol-stack and which are resolved sequentially. Consequently, succeeding tasks cannot be started before the termination of the previous.

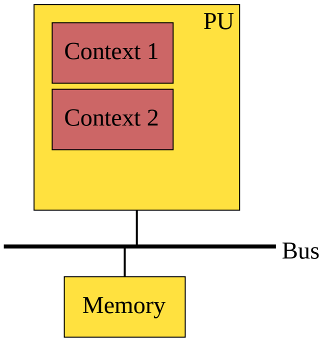


Figure 2.13: 2-way Multithreading implementation

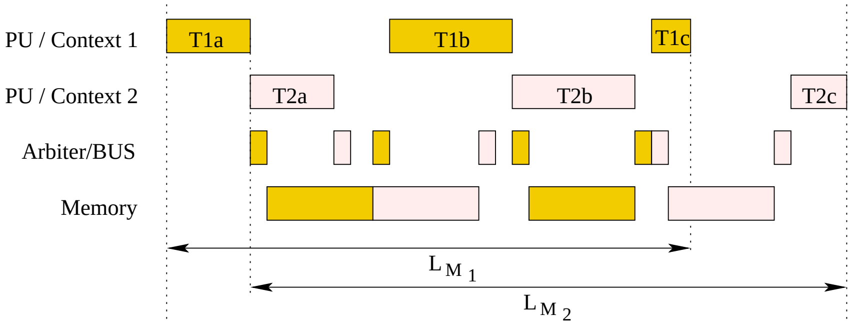


Figure 2.14: 2-way MT processing flow

The packet-processing latencies L_{M_1} and L_{M_2} consist of accumulated execution times in processing units (PU) as well as arbitration delays of shared resources and memory (MEM) access times. While the throughput is increased compared to scalar processors, MT with one functional unit does not have fundamental impacts towards reduced processing delays [75] [76].

In order to obtain a substantial reduction of protocol processing latency compared to the MT processor implementation above, two conceivable alternatives are depicted below. It should be noted that both have an additional effort in terms of system resources.

The multiprocessor (MP) system in Figure 2.15 consists of two two-way super-scalar CPUs. The two pipelines of each processor are noted as pipe a and b.

The processing flow of two packets is shown in Figure 2.16.

After the reception of a packet, e.g. packet 1, both corresponding tasks T_{1a} and T_{1b} are started in parallel. It is still assumed that task T_{1b} depends on data that result from execution of T_{1a}. Consequently, the execution of T_{1b} is started with assumed input values. The last task T_{1c} terminates earlier than in case of the 2-way MT implementation

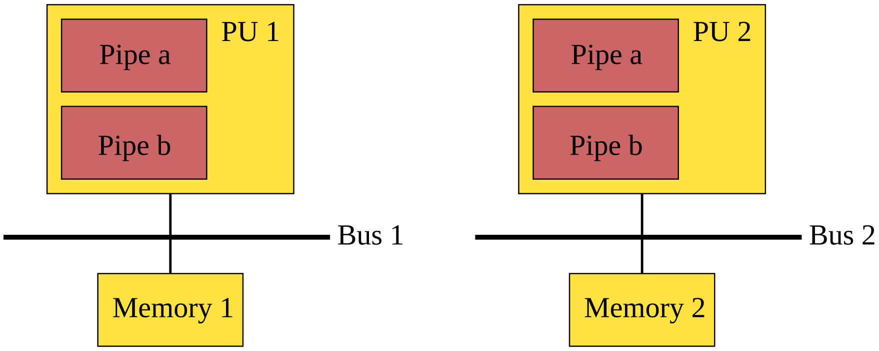


Figure 2.15: Super-scalar processor implementation

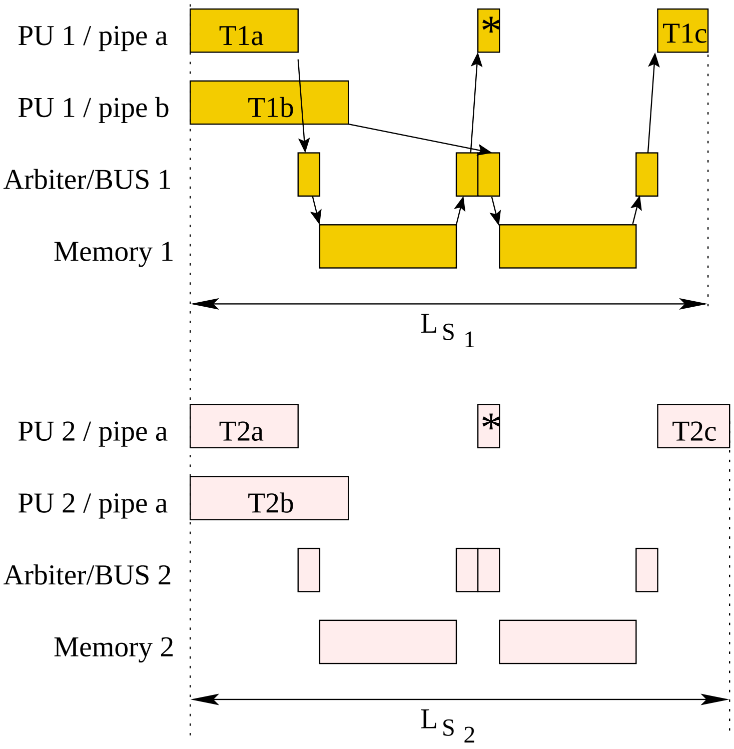


Figure 2.16: Processing flow of a super-scalar processor implementation

above. Thus, a significant reduction of L_{S_1} can be achieved, if (1) data dependencies of succeeding tasks of one thread could be resolved at the beginning of processing, and (2) the protocol-processing of a packet has a non-negligible compute-bound part. An additional effort for the verification of the result assumption, which is indicated by stars, has to be spent.

A second alternative, which exploits MT, is shown in Figure 2.17.

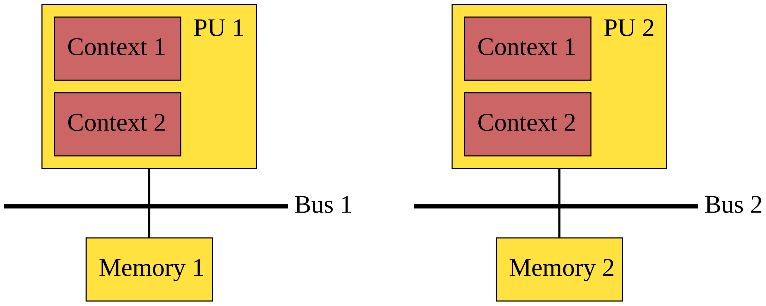


Figure 2.17: Multithreading Implementation (2 functional processing units)

It is assumed that both MT processors support each 2 contexts. Both processors have access on separate busses and memories. The flow of packet-processing in the system can be seen in Figure 2.18.

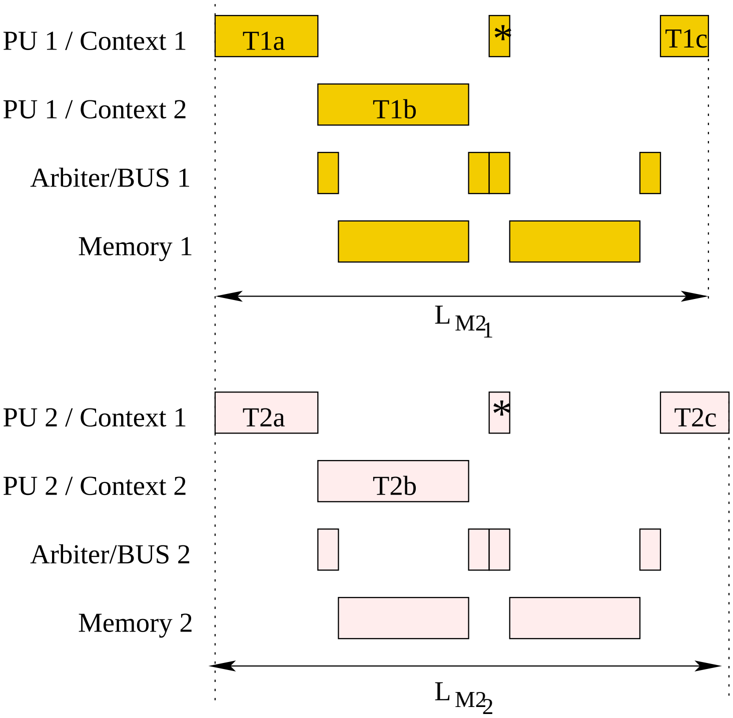


Figure 2.18: Processing flow of a MT implementation (2 functional PU)

To each PU a single packet is assigned. The context switch from T_1a to T_{1b} is done conditionally, i.e. the result of T_1a is assumed to be known at this time. The verification whether this assumption was done correctly is checked by the additional task highlighted by a star. The completion of processing for a packet is done earlier than in case of the single MT processor above. Consequently, the processing delay can be reduced by a increased amount of system resources.

Summarized, processing latencies of the last two system architectures are lower than the processing latency of a single MT network processors.

2.5 Processing flow

The delay for passing the complete packet processing chain is mainly influenced by the processing model in terms of task order.

Network processors commonly implement a sequential flow of tasks that refer to a single packet. Figure 2.19 illustrates the reference protocol-stack implementation of the IXP 1200 [23].

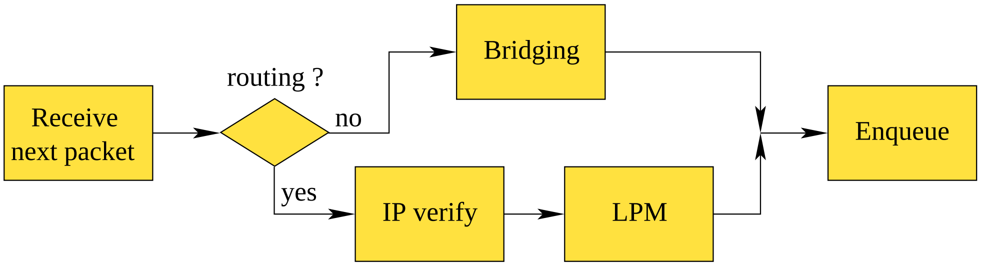


Figure 2.19: Sequential processing flow IXP 1200

The processing order follows a *sequential* flow of tasks. After the reception of a packet, either bridging has to be done or routing. In case of routing, the longest prefix match (LPM) is done after the verification of the IP header. At the end, the packet is enqueued until transmission. It should be noted that each of the blocks in the figure consist of several processing tasks.

The essential drawback is the latency which is the sum of the single processing delays. An alternative model for the task processing order implements *pseudo-parallelism*. After the termination of conditional functions within the packet processing chain, dependant tasks are started. In order to apply pseudo-parallel execution, parallel processing units are required. The presence of parallel processing resources can be used to apply the model as a reference model for evaluation purposes.

2.6 Predictive processing models

In the area of data processing, *prediction* is most commonly used to avoid latency penalties and to improve system performance. The prediction of values or events makes it possible to speculatively overcome data- or control- dependencies. The objective is to accelerate process execution through a process invocation, which is initiated earlier.

However, an uncertainty is inherent to the concept in case of occurring mispredictions. In other words, the assumed value or event may not correspond to the actual one. In this case, speculatively executed processes have to be suspended and obtained results have to be dropped. The control-path or data-path which is required to be taken instead, has to be followed additionally. In general, prediction can be divided into control-path prediction and data value prediction [77].

A brief introduction explains the fundamentals of data prediction. As a common application field of predictive processing models, branch prediction schemes used in microprocessors will be explained subsequently.

2.6.1 Fundamentals

Prediction is defined as a statement about one or more future events of a process, which are based on [78]:

- a theory
- observations

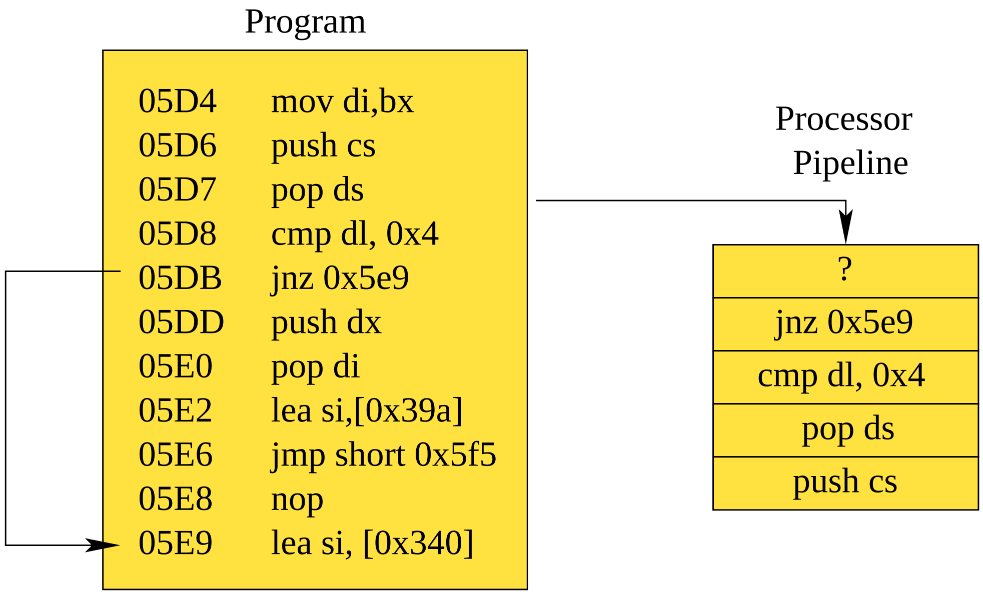
The first condition requires a statement of grounds for the prediction and a specification of premises under which the prediction can be applied. The second condition ensures that every prediction is based on an analysis of the past, i.e. is empirically funded.

A prediction which is obtained by analysis or observations from the past is only useful, if the time stability hypothesis is valid, i.e. if statistical characteristics of the process are time invariant.

2.6.2 Data prediction

Prediction is applied in different application fields. Some examples are file access prediction [79], message prediction in workstation clusters [80], as well as microprocessor branch prediction.

For the last, program branches impede machine performance though conditional branches. Continuation of program execution is not possible until a condition is not resolved and the



```
05D4    mov di,bx
05D6    push cs
05D7    pop ds
05D8    cmp dl, 0x4
05DB    jnz 0x5e9
05DD    push dx
05E0    pop di
05E2    lea si,[0x39a]
05E6    jmp short 0x5f5
05E8    nop
05E9    lea si, [0x340]
```

Figure 2.20: Program execution in microprocessors

program target address is calculated [81]. Figure 2.20 illustrates these difficulties which might occur in microprocessors during program execution.

The decision whether a jump at address 0x05DB has to be done or not, depends on the results of the previous compare instruction. Consequently, the fetch operation for the instruction register has to wait until the previous instruction has reached the last pipeline stage. As processor pipelines get deeper or issuing rate gets higher, penalties imposed by branches get larger. One way to reduce this processing penalty is to predict whether a branch will be taken or not. The corresponding address of the next instruction can be derived and pre-fetching, decoding and execution of it can be done earlier. However, in case of a branch misprediction, i.e. the predicted result of the conditional branch was incorrect, the partially processed instructions after the condition have to be dropped. Subsequently, the program execution is continued at the correct address. Summarized, the objective of branch prediction is to achieve a penalty reduction of program execution time.

In order to reach increasing values for the hit accuracy, various dynamic prediction schemes have been developed [82] [83] [84]. For example, Yeh and Patt proposed a two-level adaptive branch prediction scheme. The average prediction hit accuracy of it achieves 97 percent [85].

It should be noted that research of branch prediction was experimentally driven at the beginning. In other words, little effort was spent to analyze fundamentally the characteristics which enable an accurate prediction. Instead, most effort has been focused on an attainment of prediction schemes with high accuracies without disposing of exact models for the instruction flow. Later, analysis has been done to derive models for microprocessor program execution. For example, Evers demonstrated that branch correlation is an essential reason for enabling prediction [86].

Value prediction is another methodology for enhancing microprocessor performance. It contains a prediction of output values for operation at run-time. As a consequence, these predicted values are used to trigger execution of subsequent data-dependent instructions speculatively. By use of a hybrid value predictor, as presented in [87], a prediction hit rate of 83 percent can be achieved.

2.7 Network Traffic

Network simulations and characterizations of communication networks target on the knowledge of network traffic behavior to dimension efficiently networks and network devices [88]. An example for the exploitation of this knowledge is Internet traffic prediction, which can be used to predict the behavior of packets that are sent through communication networks [89].

Network characteristics

Extensive research has been done to understand the nature of network traffic. An essential result was the observation of *self-similarity* in various Internet traffic measurements [90] [91] [92].

Self-similarity is used for processes which show scale-independent unchanged correlational structure. In other words, the appearance is unchanged regardless of the scale at which it is viewed.

Self-similarity is closely associated with *heavy-tailed distributions*. A distribution is heavy-tailed if

$$P[X \geq x] \sim x^{-\alpha}$$

with $x \rightarrow \infty, 0 < \alpha < 2$. The expression heavy-tailed results from extremely large values with non-negligible probability [93]. Such distributions decline via a power law with small exponent, i.e. less than 2. As a consequence, when $\alpha < 2$, the random variable shows infinite variance.

Self-similar processes can exhibit *long-range dependency*. A process with long-range dependency is defined by an autocorrelation function of

$$s_{xx} \sim x^{-\beta}$$

with $x \to \infty, 0 < \beta < 1$. Further fundamentals about long memory processes can be found in [94].

Summarized, some essential observations concerning network traffic characteristics are listed in the following. However, the exact reasons for these phenomenons are not entirely determined.

- According to [95], burst sizes of FTP transfers and Telnet inter-arrival times follow a heavy-tail distribution.
- Studies in [96] show that transmission and idle times of Ethernet traffic between source and destination pairs are heavy-tailed.
- The reason of heavy-tail distributions for WWW traffic is explained by larger document sizes of WWW documents compared to UNIX files, e.g. text files or executables [90]. Furthermore, due to user behavior, heavy-tailed distributions can be found in the durations of HTTP sessions, i.e. the behavior of WWW application, and in interarrival-times of HTTP requests.
- In [97], self-similarity within network traffic is explained by TCP congestion control mechanisms: Slow-start-stage, congestion-avoidance-stage and exponential-back off-stage.

Statistics

Measured data of the Internet leads to a classification of network traffic into routing data and user data. While the former is necessary for network administration, the latter transmit application data for the network source to the destination. Several measurements have been done to understand the routing behavior in the Internet [98] [99]. However, the obtained results are not disclosed due to confidentiality reasons. Some publicly available statistics are presented in this section.

A traffic statistic gathered by MCI Worldcom, is shown in Table 2.1.

Protocol	Percentage
TCP	90
UDP	5 - 10
ICMP	1 - 3
other	0 - 4

Table 2.1: Protocol breakdown MCI Worldcom 1997

The data refer to measurement of the Internet backbone performed in 1997 [100]. Variations of the statistical distribution results from variations during daytime. Other network layer protocols, e.g. IPv6 and encapsulated IP (IP-in-IP), show an almost negligible percentage. The most frequently used Internet applications are Web traffic, DNS, SMTP, FTP, NNTP and Telnet. The majority of the applications relies on TCP as transport layer protocol. TCP traffic comes along with a significant traffic for an exchange of acknowledgment packets which can be seen in the table above.

A more recent traffic statistic can be seen in Table 2.2.

Protocol	Percentage
TCP	82 - 90
UDP	8 - 16
ICMP	1.5 - 3
other	0.5

Table 2.2: Protocol breakdown Sprintlab 2002

The results are taken from the IP Monitoring system which is deployed in the Sprint Tier-1 IP backbone network. The data are monitored at different Points-of-Presence (POP) in the USA. The reasons which have an impact on the variation from the first statistic in Table 2.1 to the second in Table 2.2 are shown below [101] [102].

- *Elephants-and-mice phenomena*: New applications lead to a statistical distribution which is characterized by a small number of high-volume streams, so-called ”elephants”, and many low-volume traffic streams, ”mice”. While the former often refers to streaming media applications, the latter results from TCP acknowledgment packets.
- *New Applications*: On some links, traffic is composed of more than 60 percent of new applications. Examples are distributed file sharing (Napster, Gnutella, etc.) and streaming data (Realaudio, Windows Media Player, etc.).
- *Firewalls*: Common firewall configurations encourage rather TCP/HTTP than UDP for real-time applications.
- *POPs*: Application access differs significantly at POPs due to local arrangements of servers, e.g. Content Delivery Networking servers.

2.8 Summary

In this chapter, packet processing methodologies have been addressed. Some of them are currently applied to network processors. The main target of the concepts is to achieve a high throughput and to optimize the system load.

The properties have been reviewed with regard to their benefits. It has been observed that the TCP/IP protocol suite displaced dedicated protocol specifications, e.g. XTP and VMTP. The complex structure of the Internet furthermore eliminates an efficient application of adaptive protocol-stacks.

The performance of protocol implementations considerably depends on the character of the performed function, i.e. memory-bound or compute-bound. In order to cope with these functions and to provide an efficient packet-processing, several concepts have been proposed. Multiprocessing and multithreading became nearly ubiquitous in state-of-the art network processor. The former provides multiple execution units to allow a simultaneous processing of packets. The latter hides access times for memory-bound functions through the support of multiple program contexts in a single microprocessor. Both concepts enable a significant increase of system throughput. The fact that both concepts provide flexibility in terms of software programmability furthermore intensifies their benefit.

The evaluation of the methods has shown that the task processing-flow causes a conceptual limitation of the processing delay. The main reason is a sequential order for the processing of tasks, such as forwarding, classification and queuing, of packets. Alternative processing models were taken to demonstrate the inherent drawback of this traditional processing flow within different implementations.

The commonly applied processing flow in NPs has been studied. A pseudo-parallel model that can execute tasks in parallel processing units possesses the capability to reduce the delay for passing the packet-processing chain.

Finally, a brief introduction in data prediction highlighted the benefits of methods which exploit value assumptions that are required for continuation of data processing.

Chapter 3

Speculative protocol-processing concept

3.1 Introduction

In order to address the delay limitation mentioned in the previous chapter and consequently to achieve lower values for the processing latency, a dyadic concept for protocol processing is proposed in the following. It consists of:

- Protocol-Stack Prediction
- Speculative Packet Processing

The targeted reduction of latencies is possible through an early resolution of data dependencies which are inherent in data packets. The concept adapts a speculative processing scheme to a networking environment. Figure 3.1 shows the processing flow of the concept.

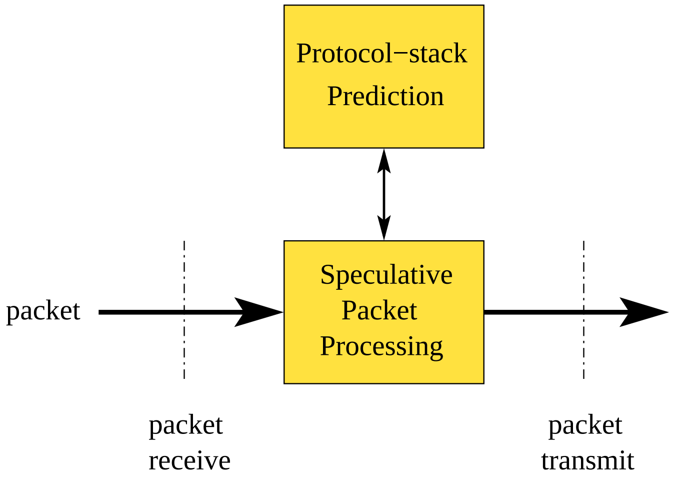


Figure 3.1: Processing concept

Protocol-stack prediction delivers an information that represents an assumption concerning the protocol-stack of a packet. The protocol-stack defines the procedure how a received packet has to be handled, e.g. if the packets has to be forwarded to another network node or has to be dropped in case of corrupted packet data. The prediction furthermore indicates the types of protocol layers and if optional fields are part of the packet header. The speculative packet processing derives the concrete processing flow as well as the corresponding tasks thereafter and starts their execution. After completion, the actual protocol-stack is shown to the prediction unit for update purposes.

3.2 Protocol-stack prediction

Protocol-stack Prediction has the objective to correctly predict the protocol-stack characteristics of packets which will be received next by the system. The protocol-stack represents a hierarchy of protocols which work together to provide required services on a communication network.

Routers comprise the physical layer, the data-link layer and the network layer. If QoS is supported, differentiated packet or traffic flow handling might be required. In this case, routers could be extended towards coverage of the transport layer as well. The prediction scheme is depicted in Figure 3.2.

A predefined number of packets p_k with $k \in \{-n, ..., -1, 0\}$, which were received earlier, are monitored and parts of their PCIs which are relevant for the prediction are gathered. According to the preselected prediction algorithm, the assumed protocol stack characteristics of the packet p_1 received next can be concluded. Thus, input for the algorithm are the

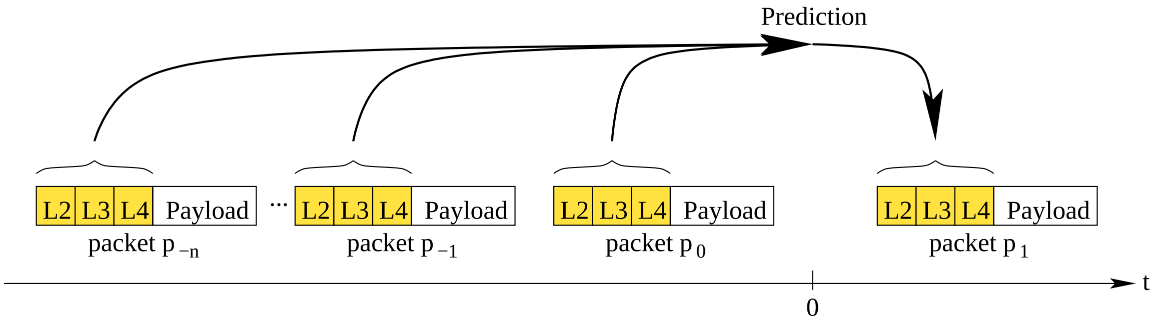


Figure 3.2: Protocol-stack prediction scheme

headers of layer 2 (L2), layer 3 (L3) and layer 4 (L4). It should be mentioned that the layers of the TCP/IP suite do not exactly correspond to the layers of the ISO reference model of communication. However, it is assumed that IP refers to layer 3 and TCP/UDP refers to layer 4.

As mentioned in the introduction, the prediction is not merely restricted to a statement that refers to the protocol types of individual layers. It encompasses furthermore the complete processing flow of a current packet as shown in Figure 3.3.

The analysis of PCI layer 3 cannot be started until its type is derived from the PCI of the lower layer, i.e. layer 2. This can be done by a derivation of the position for the corresponding layer type field in PCI layer 2. The content of the field is extracted and compared with a list of supported protocols by the system. The necessary effort for these operations is limited. Consequently, a prediction benefit in terms of processing time would be negligible if merely the layer fields would be extracted. However, several essential considerations have to be done which have an impact on the complexity of protocol-stack analysis:

- The decision whether a principal necessity for the content of layer 3 is given depends on results of several tasks. If a packet should be corrupted due to transmission failures the analysis of the PCI layer 2 locates the error. Consequently, the packet has to be dropped instead of being forwarded to layer 3 processing. Similar to this, the next hop calculation, i.e. the determination of the network path towards the packet destination and the physical interface, is only needed if the target address of layer 2 is not equal to the local address, i.e. the packet is dedicated to this local device.

- The analysis of PCI upper protocol-layer fields considers merely the type of the upper protocol layer. Consequently, the derivation of PCI upper protocol-layer fields have to be done in several stages.

- Optional PCI fields have an impact on the length of the complete protocol-layer

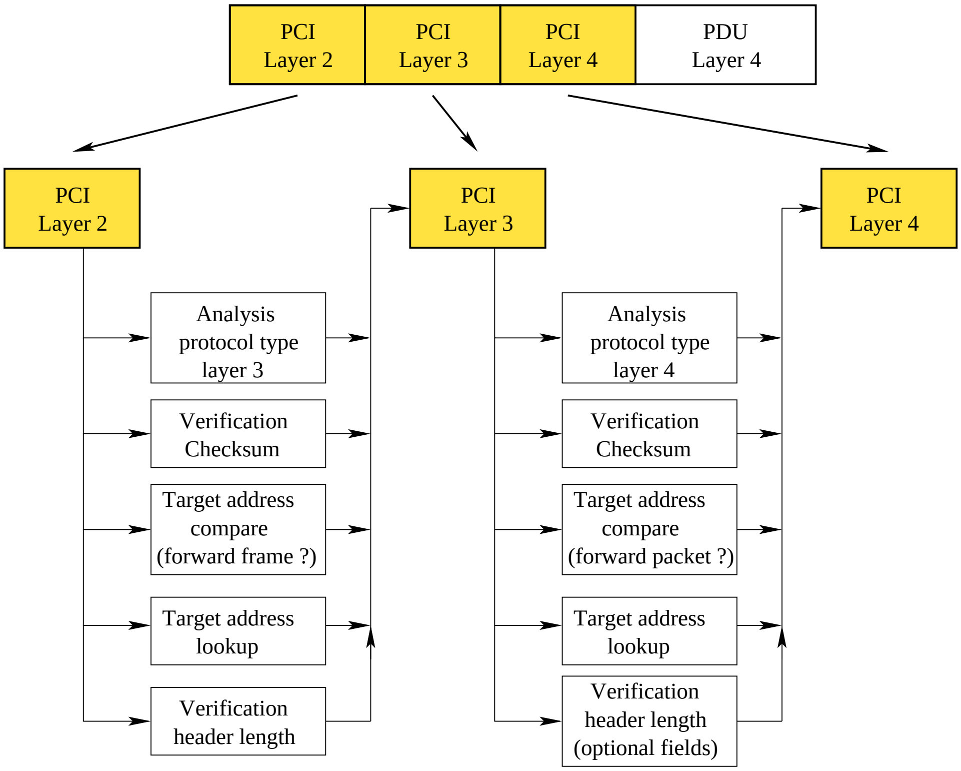

PCI Layer 2
PCI Layer 3
PCI Layer 4
PDU Layer 4
PCI Layer 2
PCI Layer 3
PCI Layer 4
Analysis protocol type layer 3
Verification Checksum
Target address compare (forward frame ?)
Target address lookup
Verification header length
Analysis protocol type layer 4
Verification Checksum
Target address compare (forward packet ?)
Target address lookup
Verification header length (optional fields)

Figure 3.3: Coverage of protocol-stack prediction

header. The offset that indicates the beginning of the upper layer PCI can be derived not until the presence of optional PCI fields has been checked.

Thus, time consuming tasks which have to be done between the analysis of upper layer PCI fields prevent from a fast derivation of the protocol-stack.

Data prediction follows a predefined algorithm that commonly uses frequencies of events that occurred before. In case of protocol-stack prediction, frequencies of received packets are used as input for the computation of prediction values. A common principle that is applied relies on the *most-frequently used* (MFU) event. A MFU model can be characterized by an output value that corresponds to the highest values of a set of event frequencies. It is applied for example in one-level branch prediction schemes [103] [104].

The applied prediction covers the network and the transport layer. The type of protocol layer two is used as an input for the prediction, i.e. a prediction is made under the condition of a certain type of protocol layer two.

As mentioned above, protocol types may provide optional header fields beyond. Considering the header encapsulation of higher layer PCIs, header field boundaries and offsets consequently depend on the presence of these optional fields. In order to provide a prediction nevertheless, an indication flag could be defined to extent the prediction to a coverage of optional fields.

In order to apply a prediction scheme for a flexible networking system environment and to provide an information that specifies i.e. protocols, daytimes, etc., statistical characteristics of the observed process have to be time-invariant. Thus, it has to be ensured that the characteristics of a process in the past will remain valid in the future. This premise is called *time stability hypothesis*. Studies about network traffic characteristics observed Ethernet traffic [91], wide area network traffic [95], WWW traffic [90] and TCP traffic [93] [97]. The results show a correlative behavior and heavy-tail distributions [94]. However, under consideration of these observations, a pragmatic starting point allows to perform a prediction nevertheless. It has to be possible therefore to assume that the time stability hypothesis is *basically* satisfied [78]. Considering the results of [101], this assumption has been made.

The prediction module has furthermore the requirement to adapt its output to variations of network statistics, e.g. during daytimes. It is furthermore assumed that the time stability hypothesis is valid for these limited periods of time, i.e. that statistical characteristics are time-invariant during this period. The integration of a dynamic behavior for the prediction enables thereafter a continuous adaptation to changed network traffic statistics.

In order to enable a dynamic adaptation of the prediction, it has to be ensured that a modified traffic distribution leads to a modified protocol-stack prediction. Parameters that determine the prediction values are the packets which have been received before. The amount of input data depends on the prediction algorithm. If the time when a packet was received is registered too, the amount of input data can be defined by a sliding window [55]. If the time stamp of packet reception is not kept, the variability can be realized by a cyclic mechanism that periodically resets the data base of the computation data for the prediction.

Performance constraints that refer to certain protocol types can require a favored processing, e.g. real-time traffic which commonly relies on UDP for the transport layer. In order to cope with this demand, a weight factor can be defined to assign a higher prediction priority. This means that the calculation of a predicted protocol-stack considers the frequencies of received packets as well as the priority weights for the protocol-stacks.

In summary, the goal of the prediction is to provide an information that specifies the protocol-stack of a next packet. This data determines the processing order and the particular tasks which have to be performed with the packet.

3.3 Speculative protocol-processing

The objective of *speculative protocol processing* is to perform the entire protocol-processing of a packet in a shorter period of time, compared to traditional methods. The approach is based on the fact that required input data are not entirely known at the initiation of task processing. The unknown data and flag, e.g. protocol layer types and flags concerning optional fields, are derived from the protocol-stack prediction.

The order of processing depends on several factors, e.g. the protocol types and supported applications such as DiffServ, MPLS, IPSec, VPN. In addition, due to performance requirements, execution order can differ at the network edge, in the core and in termination nodes. In Fig. 3.4, the flow of speculative data packet processing is illustrated.

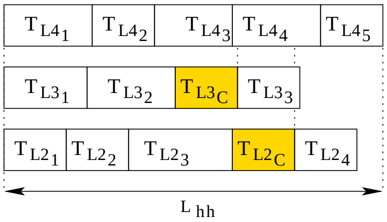


Figure 3.4: Speculative Packet Processing (Layer 2 and 3 hit)

Tasks T_{Li_j}, which refer to one protocol layer L_i with $i \in \{2,3,4\}$, are conceived to be processed sequentially. The order of execution is expressed by increasing values of j. The execution of tasks in a speculative processing model depends on the availability of parallel processing resources.

In case of speculative processing, execution of first tasks for protocol layer 2, 3 and 4 , i.e. T_{L2_1}, T_{L3_1} and T_{L4_1}, are initiated simultaneously. Each layer possesses one task, which deals with the determination of the protocol type for the next upper layer. Further on, this task is called *checktask*. Checktasks of layer 2 and 3, i.e. T_{L2_C} and T_{L3_C} are highlighted in grey.

After completion of a checktask, the actual protocol layer type of the current packet has been analyzed. This resolution of the data dependency that concerns the type of the higher protocol-layer enables the partial verification of the performed prediction. This is illustrated by dashed lines. At this point, two possible cases can occur:

1. Prediction hit
2. Misprediction

In order to avoid misunderstandings, it should be emphasized that protocol stack prediction delivers one single information that identifies the complete stack. Thus, the prediction delivers an assumption for layer 2 *AND* higher layers. By contrast, checktask resolution merely serves to evaluate the prediction statement with respect to one layer.

In case of a prediction hit, it can be concluded that control information concerning the protocol layer has been delivered correctly by the prediction. Tasks, which have been initiated earlier, were based on a correct input. Consequently, results which have been obtained earlier, can be kept. The execution of simultaneous tasks can proceed. Finally, remaining tasks are started subsequently.

Otherwise, i.e. in case of misprediction, the assumption of a value, which identifies the higher layer protocol type, was false. This means, instructions that have been already performed, were founded on a wrong assumption concerning the packet header type. As a consequence, results which were obtained have to be dropped. Furthermore, tasks which are executed simultaneously to the checktask have to be aborted. In order to finish packet processing, the correct information that identifies the upper layer type is derived from the checktask and processing of the corresponding layer is restarted upon correct input data. In Figure 3.5, these tasks are marked with stars $T^*_{Li_j}$. Even if partial results of higher protocol layers might be correct, an individual verification does not take place. All results of higher layers are dropped instead to avoid an additional computation complexity. An additional processing effort is necessary for the derivation of the actual tasks which have to be processed instead the mispredicted tasks. This effort in terms of time is referred to as the additional processing time (APT). Reasons for a value $APT \neq 0$ might be memory table lookups or a clearance of processor pipelines.

The checktask T_{L3_C} similarly evaluates the prediction concerning layer 4. Depending on the correctness of the prediction, result values are kept or dropped and processing of layer 4 is proceeded or restarted.

In case of a correct prediction for both layers, layer 3 and 4, the complete processing time is called the hit latency L_{hh}. “h” signifies a prediction hit. While the left index refers to protocol layer three, the right corresponds to layer four.

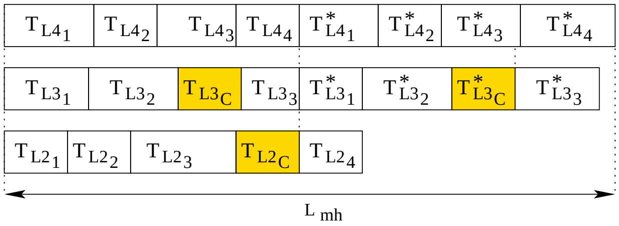


Figure 3.5: Speculative Packet Processing (Layer 2 miss / Layer 3 hit)

The figure above shows that it can be conceived to extend the scheme towards multi-level prediction. After verification of protocol layer 3 and in case of failure of the *first* prediction, a second protocol stack prediction can be requested. The second prediction is based on the present knowledge of layer 3. Thus, this prediction stage comprises a statement about protocol layer 4 in conjunction with the already known lower layers.

This implies that an uncertainty of an incorrectly predicted layer 4 persists until $T^*_{L3_C}$ has been processed. In the figure above, dashed lines illustrate the time, when results of checktask calculation are given.

In case of an incorrect first and a correct second prediction, the processing latency is L_{mh}. “m” signifies a misprediction.

Figure 3.6 pictures a protocol-processing flow in case of two mispredictions. The processing delay is L_{mm}.

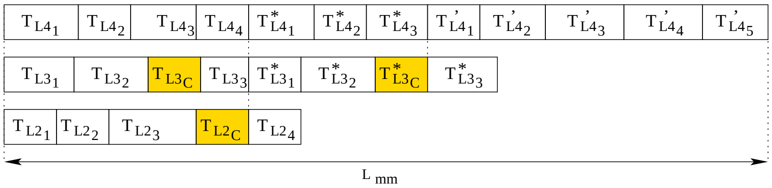

T_{L4_1} T_{L4_2} T_{L4_3} T_{L4_4} $T^*_{L4_1}$ $T^*_{L4_2}$ $T^*_{L4_3}$ T'_{L4_1} T'_{L4_2} T'_{L4_3} T'_{L4_4} T'_{L4_5}
T_{L3_1} T_{L3_2} T_{L3_C} T_{L3_3} $T^*_{L3_1}$ $T^*_{L3_2}$ $T^*_{L3_C}$ $T^*_{L3_3}$
T_{L2_1} T_{L2_2} T_{L2_3} T_{L2_C} T_{L2_4}
L_{mm}

Figure 3.6: Speculative Packet Processing (Layer 2 and 3 miss)

The fourth case that can occur is a correct prediction of layer three and a wrong for layer 4. Hereby, no second prediction concerning layer 4 has to be requested. The obtained latency is L_{hm}.

In order to illustrate the difference of the proposed concept and traditional methods, Figure 3.7 (a) shows the sequential processing flow of protocol processing and (b) the pseudo-parallel flow.

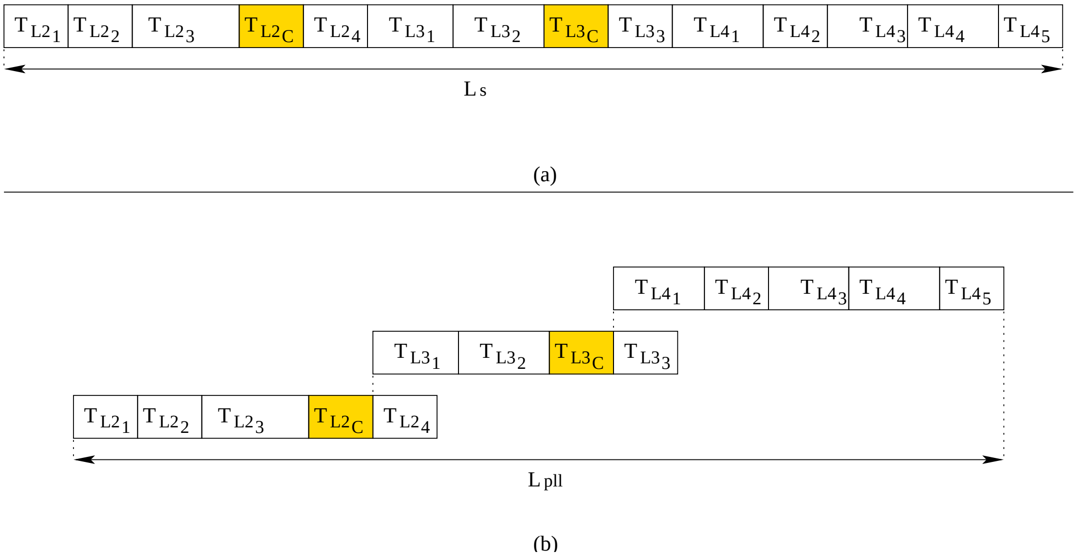

T_{L2_1} T_{L2_2} T_{L2_3} T_{L2_C} T_{L2_4} T_{L3_1} T_{L3_2} T_{L3_C} T_{L3_3} T_{L4_1} T_{L4_2} T_{L4_3} T_{L4_4} T_{L4_5}
L_s
(a)
T_{L4_1} T_{L4_2} T_{L4_3} T_{L4_4} T_{L4_5}
T_{L3_1} T_{L3_2} T_{L3_C} T_{L3_3}
T_{L2_1} T_{L2_2} T_{L2_3} T_{L2_C} T_{L2_4}
L_{pll}
(b)

Figure 3.7: Sequential and pseudo-parallel processing

The corresponding latencies are L_s in case of sequential processing and L_{pll} in case of pseudo-parallel processing.

3.4 Concept benefit

According to the given prediction, one of four values for the latency can be obtained, i.e. L_{hh}, L_{hm}, L_{mh} or L_{mm}. For a period of time and a sample of N packets, the mean processing latency $\overline{L_N}$ is

$$\overline{L_N} = \frac{1}{4}\left[\frac{1}{N_{hh}}\sum_i L_{hh_i} + \frac{1}{N_{hm}}\sum_j L_{hm_j} + \frac{1}{N_{mh}}\sum_k L_{mh_k} + \frac{1}{N_{mm}}\sum_l L_{mm_l}\right] \tag{3.1}$$

with the number of packets N_{hh} in case of layer three hit / layer four hit, N_{hm} in case of layer three hit / layer four miss, N_{mh} in case of layer three miss / layer four hit and N_{mm} in case of layer three miss / layer four miss. Packet numbers for the four possible case are indicated by i, j, k and l.

In case of mispredictions, processing delays have equal or higher values than in case of correct protocol-stack assumptions. The essential benefit of the speculative processing method is a mean processing delay L_N which possesses a lower value than alternative processing models, e.g. sequential or pseudo-parallel processing.

The proposed concept is not restricted concerning systems where it can be applied. A usage in network nodes, e.g. routers, can be aspired to as well as an integration in network terminations, i.e. host computers.

Chapter 4

Abstract concept evaluation

4.1 Introduction

The processing model which was introduced in the previous chapter was motivated by the demand for short processing latencies in network nodes. The evaluation of the speculative concept will be done in two stages. This procedure is inspired by the ultimate goal, which is the evaluation of the speculative processing concept and not the development of an optimized architecture for the concept. The first stage consciously extracts details and constraints which are specific for an architectural implementation, e.g. access methods to shared system resources and network protocols. The main intention of this evaluation part is to derive parameters which are generally bound to the concept and not to its architectural implementation and to determine their quantitative impact on the processing performance. This will be the subject of this chapter. Succeeding chapters are concerned with the second evaluation stage. There, the concept will be implemented in a system architecture and the impact of added details will be analyzed with respect to performance and workload.

The first evaluation stage allows to obtain an estimation of the order of magnitude for relevant performance metrics of the processing concept. For each concept-relevant parameter, an evaluation scenario is worked out. The separate scenarios are analyzed to reflect the particular impact of the corresponding parameter on the system behavior. The realization of the concept evaluation is done upon an analytical study and not a simulation. During this analysis other parameters are set to fixed values.

The quality of the concept is expressed by two relevant performance metrics, namely *latency reduction* and *additional costs*. The latency reduction measures the processing latency of the speculative concept in comparison to the reference model of the analysis. Additional costs represent the additional workload of processing resources which perform speculative processing in comparison to the reference model. It should be noted that no implementation costs of processing resources are meant. Both metrics are defined as relative quantities that refer to a reference system. For this purpose, the pseudo-parallel processing model, which has been mentioned in chapter 2, will be used. It possesses parallel processing capabilities and differs from the speculative approach in the initiation time of task execution, i.e. the execution of a specific layer's task is not started until corresponding data-dependencies are resolved, and the lack of a prediction.

4.2 Evaluation

4.2.1 Predefinitions

Tasks and layers

In order to fulfill execution of a complete process flow, process tasks that belong together are joined to a unit. Each unit is assigned to one layer that comprises a well-defined service. Individual layers are grouped hierarchically as shown in Figure 4.1.

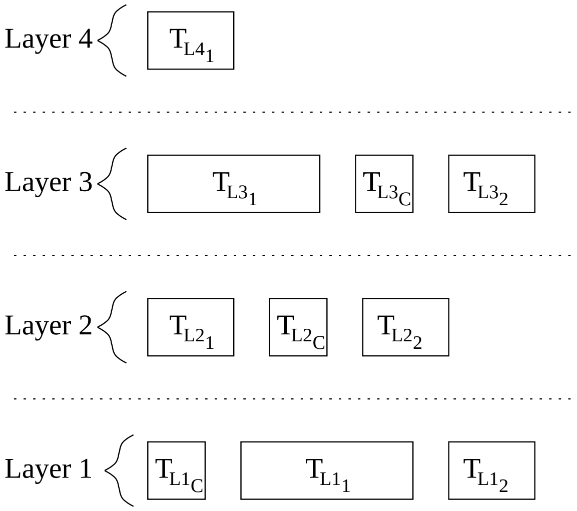


Figure 4.1: Layer hierarchy

The interaction between layers is done through interfaces. A task represents one or multiple functions to fulfill the execution of certain processing functionalities. Considering the control-path that decides the process flow, it can be distinguished between two types of tasks: *normal tasks*[1] and *checktasks*. Normal tasks are characterized by an execution that has no impact on other layers. By contrast, the execution of checktasks allows to derive types of other layers. Thus, they condition the control-path of the process. Tasks which have an impact on processing of the same layer are referred to as normal tasks. For the

[1] in the following, the expressions *function*, *normal task* and *task* are equivalent

analysis, it is defined that tasks referring to one layer are processed in a sequential order starting with task 1. The nomenclature is:

- T_{Li_j}: jth task of layer i
- T_{Li_C}: checktask of layer i

Processing models

Both models, i.e. the speculative model as well as the pseudo-parallel reference model are defined as systems that possess the capability for a concurrent execution of tasks which refer to different layers.

In order to analyze the impact of concept parameters on the system performance, the structure of tasks is restricted to a minimum number of tasks that enable a speculative processing model and allow a significant analysis.

The execution of tasks proceeds until all unfinished tasks are completed. Tasks which were speculatively processed might be dropped directly after the detection of a misprediction. The results of corresponding tasks are dropped. Then, the actual tasks have to be processed until their completion.

The lowest layer for the analysis is predefined to layer 2. The processing of tasks that refer to layer 2 is done without an input condition. It is assumed that the type of layer 2 is known before start of processing.

Reference model

As mentioned above, the pseudo-parallel model starts task processing of a layer not until the layer type is identified. This depends on the execution of the corresponding checktask. Figure 4.2 shows the reference model which contains layer two and three.

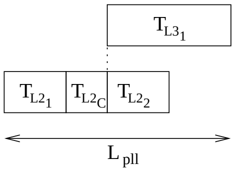


Figure 4.2: Pseudo-parallel processing implementation

T_{L2_1}, T_{L2_2} and T_{L3_1} are used as normal tasks. The result of the checktask T_{L2_C} allows to conclude on the type of layer three.

Speculative processing model

The system which is evaluated relies on the speculative processing model. Two processing cases can occur for the given task structure. Figure 4.3 illustrates a correct prediction with the prediction hit latency L_h.

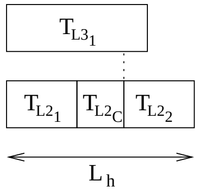


Figure 4.3: Speculative processing / case prediction hit

The time when the type of layer 3 has been analyzed, is highlighted by a dotted line. Else, Figure 4.4 depicts a prediction miss with the latency L_m.

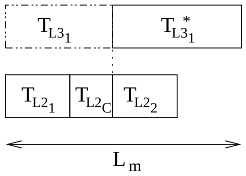


Figure 4.4: Speculative processing / case misprediction

If a misprediction for layer three is detected, the speculatively processed task T_{L3_1} is identified as a wrongly performed task. Its processing is interrupted at once and the actual task $T^*_{L3_1}$ is initiated instead. The additional effort for the conclusion on task $T^*_{L3_1}$ is neglected during this introduction of the processing models.

Metrics

The two observed metrics are:

- Reduction of processing time (latency reduction)
- Increase of processing costs (additional costs)

Latency reduction

The mean reduction of processing time $\overline{r}$ is the relation between speculative processing and pseudo-parallel processing latency. It is defined to

$$\overline{r} = 1 - \frac{\overline{L_{sp}}}{L_{pll}} \tag{4.1}$$

$\overline{L_{sp}}$ represents the mean processing time for the speculative processing and L_{pll} the processing time in case of pseudo-parallel processing. $\overline{L_{sp}}$ is the linear combination of L_h and L_m according to

$$\overline{L_{sp}} = \frac{1}{N_h + N_m}(N_h L_h + N_m L_m) \tag{4.2}$$

N_h and N_m are the numbers of correct predictions and mispredictions, respectively. The complete number of received packets is $N = N_h + N_m$.

The range of values for the reduction is $r < +1$. A value of $0 < r < 1$ indicates positive latency reduction, $r = 0$ no improvement and $r < 0$ represents a penalty in processing time.

Additional costs

In order to quantify the workload of a processing resource, “costs” are defined. The assigned value corresponds to the time when a processing resource is busy. Else, if a processing resource is idle, the costs value is set to 0. The mean *additional costs* (AC) function is defined to as the ratio of processing workload of the speculative and of the pseudo-parallel model:

$$\overline{AC} = \frac{\overline{c_{sp}}}{c_{pll}} - 1 \tag{4.3}$$

where $\overline{c_{sp}}$ represents the mean costs in case of pseudo-parallel processing and c_{pll} the costs in case of pseudo-parallel processing.

In case of pseudo-parallel processing, we have:

$$c_{pll} = \sum_{i,j} T_{Li_j} \tag{4.4}$$

with the layers i and the tasks j. Apart from numerical values, j includes c, i.e. checktasks are considered in the equation.

The mean costs $\overline{c_{sp}}$ for the speculative processing model consist of costs in case of hit prediction c_h and costs in case of misprediction c_m. They are

$$\overline{c_{sp}} = \frac{N_h}{N} c_h + \frac{N_m}{N} c_m \tag{4.5}$$

The former is

$$c_h = \sum_{i,j} T_{Li_j} \tag{4.6}$$

The latter consists of correctly and wrongly processed tasks.

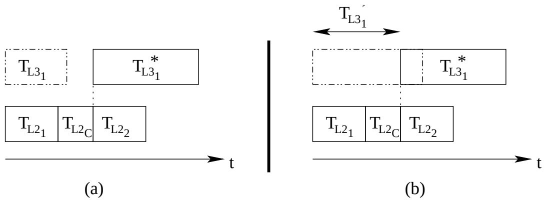


Figure 4.5: Costs in case of misprediction

In Figure 4.5, the two cases that can occur are depicted. If $T_{L3_1} \leq T_{L2_1} + T_{L2_c}$ (Figure 4.5(a)), the costs c_m can be derived as

$$c_m = \sum_{i,j} T_{Li_j} + \sum_{i,j} T_{Li_j^*} \tag{4.7}$$

Otherwise, i.e. $T_{L3_1} > T_{L2_1} + T_{L2_c}$ (Figure 4.5(b)), the costs c_m are

$$c_m = \sum_{i,j} T_{Li_j} + \sum_{i,j} T_{Li_j^*} + \sum_{i,j} T_{Li_j'} \tag{4.8}$$

c_m is composed of processing tasks T_{Li_j} that include normal tasks and checktasks as well as one or multiple actual tasks $T_{Li_j^*}$ which are executed after interruption of wrongly processed tasks. In case of a misprediction, costs of wrongly processed and aborted tasks $T_{Li_j'}$ have to be added. Furthermore, by processing more than two layers, a superposition of mispredictions and prediction hits might occur. Consequently, equations (4.6) and (4.7) have to be generalized accordingly.

Analytical model

The analytical model of the system considers six different input parameters P_i with $i \in \{1, \ldots, 6\}$ that are immanent to the speculative processing model. The parameters are listed below.

1. Number of predicted layers (P_1)
2. Additional processing time in case of misprediction (P_2)
3. Prediction accuracy (P_3)
4. Resolution time of control-flow conditions, i.e. the time when the control-flow conditions are resolved (P_4)
5. Processing time of control-flow conditions, i.e. the duration of the checktask (P_5)
6. Prediction model: single-level or multi-level (P_6)

Figure 4.6 illustrates the black box which contains the analytical model. The mean values $\overline{r}$ and $\overline{AC}$ represents the system output.

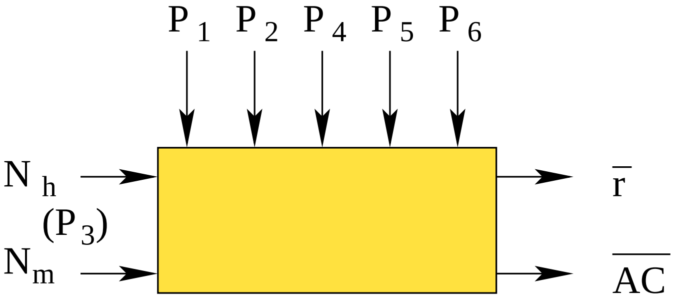


Figure 4.6: Analytical model

For the analytical model, the applied prediction is reduced to a numerical value that is defined by the fraction of correct predictions N_h and the complete number of events N. The value corresponds to P_3. The complete number of events is the sum of correct predictions N_h and mispredictions N_m. During this analysis, a time-variant behavior of the prediction is not applied.

Numerical values

Values for task execution times are chosen by estimation of their processing effort. For instance, check-tasks are assumed to be short. Considering analogies to networking, tasks of higher layers are defined to demand a higher effort in terms of time consumption [55]. During the analysis, the applied hierarchy is defined to comprise layer 2 and higher.

4.2.2 Scenarios

The number of the employed scenarios for the abstract evaluation corresponds to the observed parameters mentioned above. The impact of these parameters on latency and costs is examined in the following scenarios. For the analysis, the following assumptions have been made:

- Task execution times are set to $T_{L_{i_j}} = 10\tau$ for tasks below the highest layer, $T_{L_{i_C}} = 5\tau$ for checktasks and $T_{L_{i_j}} = 20\tau$ for the highest layer of the inspected scenario.
- The processing covers layer 2 and 3.
- The additional processing time (APT) is set to $T_{APT} = 0$.
- The processing time of task $T_{L_{3_1}}$ possesses a well-defined value. A value extension in terms of an additional processing time is expressed by δ. The default value is $\delta = 0$.
- Static values for the prediction hit ratio are 0.2, 0.5 and 0.8.
- An a priori knowledge for the type of layer 2 is assumed.

Deviations from these assumptions are explicitly mentioned in the particular scenario.

4.2.2.1 Scenario 1: Number of predicted layers

In the first scenario, the impact of predicted protocol layer is analyzed. Processing models which encompass layer 2 and 3 are depicted in Figure 4.7.

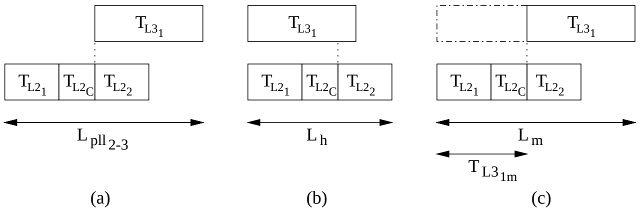


Figure 4.7: Processing flows (layer 2 - 3)

While Figure 4.7 (a) shows the sequence of tasks in the reference model, (b) and (c) illustrate the flow of the speculative model. It is assumed that the type of layer 2 is known a priori. The corresponding latencies are:

$$\begin{aligned} L_{pll_{2-3}} &= T_{L2_1} + T_{L2_C} + max\{T_{L2_2}; T_{L3_1}\} && (4.9) \\ L_h &= max\{T_{L2_1} + T_{L2_C} + T_{L2_2}; T_{L3_1}\} && (4.10) \\ L_m &= T_{L2_1} + T_{L2_C} + max\{T_{L2_2}; T_{L3_1}\} && (4.11) \end{aligned}$$

The mean latency for the speculative model finally is:

$$\overline{L_{sp_{2-3}}} = \frac{1}{N}(L_h N_h + L_m [N - N_h]) \quad (4.12)$$

Figures 4.8 and 4.9 show the processing flow up to layer 4 and 5 , respectively. The mean latencies for the speculative model $L_{sp_{2-4}}$ and $L_{sp_{2-5}}$ can be derived likewise from correctly and wrongly performed predictions.

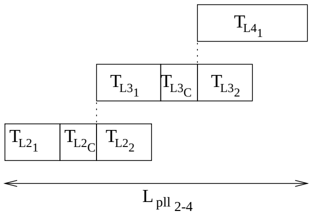


Figure 4.8: Pseudo-parallel processing flow (layer 2 - 4)

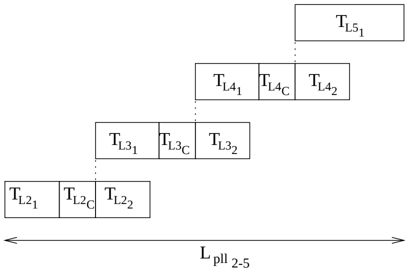


Figure 4.9: Pseudo-parallel processing flow (layer 2 - 5)

Figure 4.10 highlights that latency reduction increases with an increasing amount of predicted layers.

The ratio of correct and wrong predictions compared to the complete analysis time is expressed by the relation of correct predictions and mispredictions N_i, N_{ii} and N_{iii} with $i \in \{m, h\}$ for processing up to layer 3, 4 and 5 and the total number of events N. The

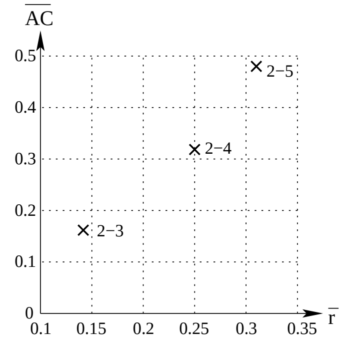


Figure 4.10: Scenario 1: $\overline{AC}, \overline{r}$ / Parameter: number of predicted layers

ratio of prediction hits and failures to the accumulated number of both is $N_i/N = 0.5$, $N_{ii}/N = 0.25$ and $N_{iii}/N = 0.125$. In case of a prediction up to layer 5, $\overline{r}$ can be reduced up to 31 %. However, it can be mentioned that additional costs increase more than $\overline{r}$. This results in a value for $\overline{AC}$ of 48 %.

The figures 4.11 and 4.12 illustrate the graphs for different ratios for correct predictions and mispredictions. While N_h/N exclusively refers to the processing structure that covers layers 2 and 3, N_{hh}/N is used for the structure up to layer 4 and N_{hhh}/N up to layer 5. Thus, it has to be mentioned that no relation is defined between these hit ratios. Figure 4.11 shows $\overline{AC}$ and $\overline{r}$ for $N_h/N = 0.5$ for layers 2 - 3, $N_{hh}/N = 0.5$ for layers 2 - 4 and $N_{hhh}/N = 0.5$ for layers 2 - 5. The remaining ratios for each processing structure have identical values. Considering network traffic statistics that allow higher values for correct prediction ratios [101], Figure 4.12 illustrates the properties for $N_h/N = 0.7$ for processing up to layer 3, $N_{hh}/N = 0.7$ for processing up to layer 4 and $N_{hhh}/N = 0.7$ if layer 5 is part of processing.

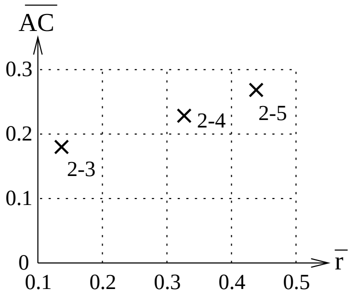


Figure 4.11: Scenario 1: $\overline{AC}, \overline{r}(N_h/N = N_{hh}/N = N_{hhh}/N = 0.5)$ / Parameter: number of predicted layers

It can be seen that an increase of correct predictions leads to a delay reduction. In parallel, additional costs are decreased due to a lower processing effort.

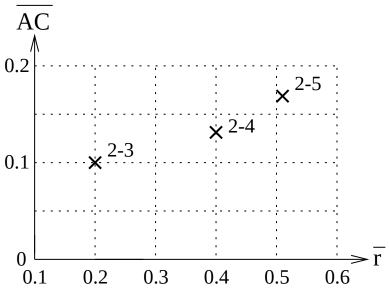


Figure 4.12: Scenario 1: $\overline{AC}, \overline{r}(N_h/N = N_{hh}/N = N_{hhh}/N = 0.7)$ / Parameter: number of predicted layers

4.2.2.2 Scenario 2: Additional processing time in case of misprediction

In case of occurring mispredictions, simultaneously processed tasks have to be aborted. The tasks which have to be processed instead have to be derived from the results of the checktasks. In order to express the corresponding additional effort, the additional processing time T_{APT} is defined. Thus, the processing delay for misprediction of layer 3 is

$$L_m^* = T_{L2_1} + T_{L2_C} + max\{T_{L2_2}; T_{APT} + T_{L3_1}\} \tag{4.13}$$

The graphs for $\overline{r} = function(T_{APT})$ are shown in Figure 4.13.

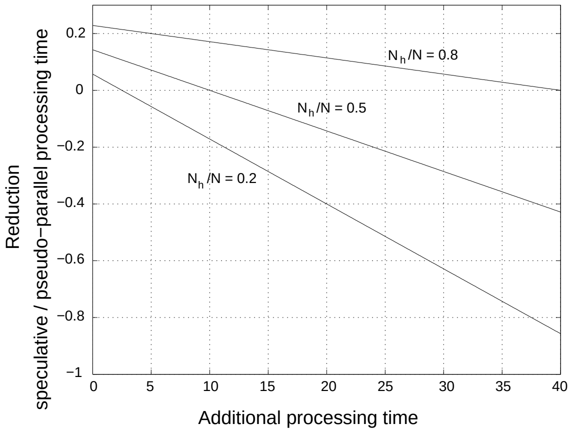


Figure 4.13: Scenario 2: $\overline{r} = function(T_{APT})$

Note that large values for T_{APT} result in an increase of the speculative processing delay compared to the reference model. Figure 4.14 shows the additional costs for different ratios of the prediction accuracy $\frac{N_h}{N}$.

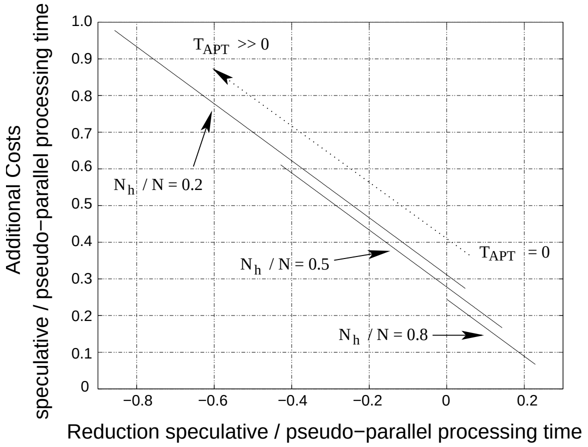


Figure 4.14: Scenario 2: $\overline{AC}, \overline{r}$ / Parameter: T_{APT}

The figure above shows that additional costs equivalently arise with an increasing value for T_{APT}. It has to be targeted that T_{APT} has to be reduced to achieve an optimized delay reduction.

4.2.2.3 Scenario 3: Prediction accuracy

The influence of prediction accuracy is herein observed. With (4.1), (4.9), (4.11), and (4.12), the reduction of processing time is given by

$$\overline{r} = 1 - \beta_1 \frac{N_h}{N} - \beta_2 = (1 - \beta_2) - \beta_1 \frac{N_h}{N} \tag{4.14}$$

with $\beta_1 = \frac{L_h - L_m}{L_{pll}}$ and $\beta_2 = \frac{L_m}{L_{pll}}$.
According to (4.9) and (4.11), we obtain

$$\beta_2 = 1 \tag{4.15}$$

and consequently

$$\overline{r} \sim \frac{N_h}{N} \tag{4.16}$$

Due to (4.5), we obtain linear behavior of additional costs

$$\overline{AC} \sim -\frac{N_h}{N} \tag{4.17}$$

Figure 4.15 shows the graph of additional costs and delay reduction for different values of N_h.

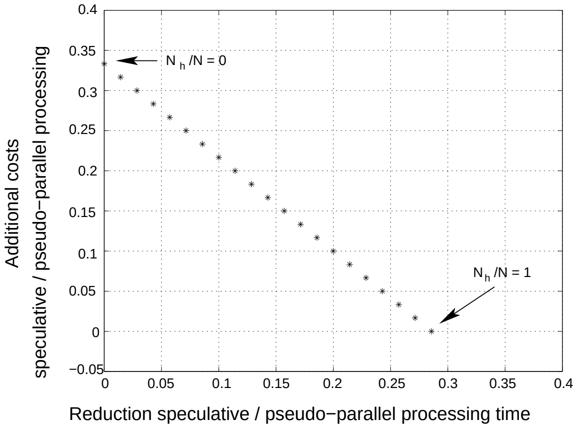


Figure 4.15: Scenario 3: $\overline{AC}, \overline{r}$ / Parameter: prediction accuracy

The processing delay reduction increases and costs decrease in case of higher values for $\frac{N_h}{N}$. If no prediction failure occurs, a maximum value for the latency reduction of $\overline{r} = 28\%$ is achieved.

In case of misprediction, an additional demand for control-flow processing might arise, i.e. $T_{APT} \neq 0$. According to equation 4.13, the processing latency in case of mispredictions extends to

$$L_m^* = T_{L2_1} + T_{L2_C} + max\{T_{L2_2}; T_{APT} + T_{L3_1}\} \tag{4.18}$$

Consequently, the reduction is less than in 4.14:

$$\overline{r^*} = 1 - \frac{L_h - L_m{}^*}{L_{pll}} \frac{N_h}{N} - \frac{L_m{}^*}{L_{pll}} \tag{4.19}$$

The assumption that $T_{L3_1} \geq T_{L2_2}$ yields

$$\overline{r^*} = \overline{r} + \frac{T_{APT}}{L_{pll}} \frac{N_h}{N} - \frac{T_{APT}}{L_{pll}} \tag{4.20}$$

Figure 4.16 illustrates the consequences of different values for T_{APT}, i.e. $T_{APT} = 0\tau$ and $T_{APT} = 10\tau$.

If the ratio for correct predictions is less than 50 %, the delay reduction becomes negative. This means that the mean speculative processing delay is higher than the processing delay of the reference model.

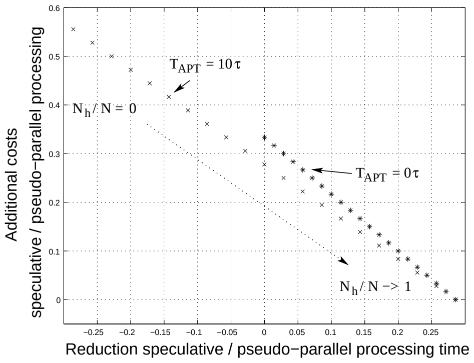


Figure 4.16: Scenario 3: $\overline{AC}, \overline{r}$ / Parameter: prediction accuracy, $T_{APT} \neq 0$)

The impact of a varied value for the execution time T_{L3_1} is observed. An additional processing time is defined by δ. It should be mentioned that δ and APT are different parameters. While the former is used for variations of T_{L3_1}, the latter refers to additional effort in case of a misprediction. The resulting processing time for the task in layer three is $T'_{L3_1} = T_{L3_1} + \delta$. Figure 4.17 illustrates the additional costs and the delay reduction for different values of δ, i.e. $\delta_1 = 5\tau$ and $\delta_1 = 40\tau$.

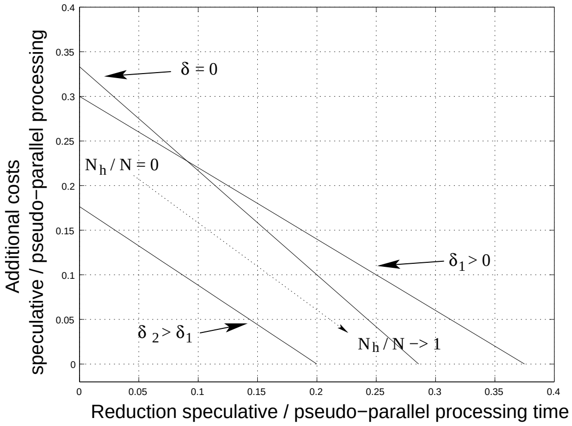


Figure 4.17: Scenario 3: $\overline{AC}, \overline{r}$ / Parameter: prediction accuracy, $T'_{L3_1} = T_{L3_1} + \delta$

It can be observed that for $\frac{N_h}{N} \rightarrow 1$ the delay reduction possesses a local maximum. The corresponding value for δ can be derived from $T_{L3_1} + \delta = T_{L2_1} + T_{L2_c} + T_{L2_2}$. For the predefined values, δ is 5 τ. This behavior results from the fact, that for $\delta < 5\tau$, δ has only an impact on L_{pll}, but not on $\overline{L_{sp}}$. For values $\delta > 5\tau$, the mean reduction $\overline{r}$ decreases.

4.2.2.4 Scenario 4: Resolution time of control-flow conditions

Now, the impact of the time is considered when the execution of the checktask has been completed. In order to reduce the complexity of the task structure, the processing model is simplified towards two normal tasks and the checktask T_{L2_C} as shown in Figure 4.18.

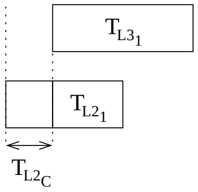


Figure 4.18: Scenario 4: Completion time of the checktask

The graphs for $N_h/N = 0.5$ and $N_h/N = 0.8$ are shown in Figure 4.19. The dotted line illustrates the graph direction for increasing values of T_{L2_C}.

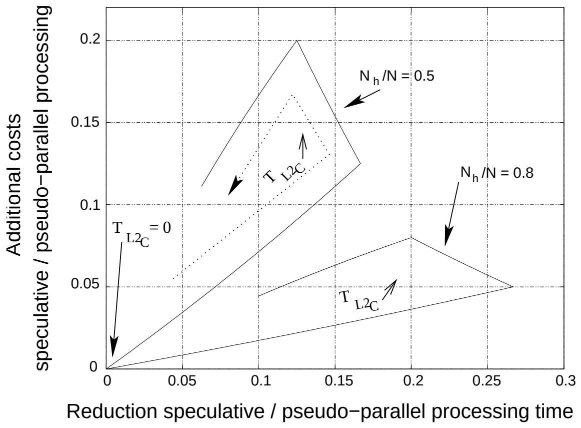


Figure 4.19: Scenario 4: $\overline{AC}, \overline{r}$ / Parameter: resolution time of control-flow conditions

Both, the reduction of mean processing delay $\overline{r}$ as well as $\overline{AC}$ possess a local maximum which result from the overlap of synchronous processing tasks. Maximum values for $\overline{r}$ are obtained if $T_{L2_C} + T_{L2_1} = T_{L3_1}$. Maximum additional costs $\overline{AC}$ are given for $T_{L2_C} = T_{L3_1}$, i.e. if a maximum contribution of mispredicted tasks T_{L3_1} occurs. The following conclusions can be drawn: For early resolution times of the control-flow condition, the main part of a speculative processing delay are time-consuming tasks of higher layers. For late resolution times, the dominant factor for $\overline{L_{sp}}$ is the processing layer two. Consequently, an maximum mean latency reduction can be achieved if the task processing order meets $T_{L2_C} + T_{L2_1} = T_{L3_1}$.

4.2.2.5 Scenario 5: Processing time of control-flow conditions

The difference to the previous scenario lies in the time when the control-flow condition is resolved, i.e. when checktask processing is completed. Thus, the impact of the position of the checktask in relation to the start of processing has been studied in scenario 4. Now, the pure processing duration of the checktask independent of its sequential processing position in the corresponding layer is observed. The computation time of the checktask has no direct relationship to the time when the checktask is invoked and when it completes processing. The processing time of the control-flow condition is considered in Figure 4.20. The non-linear behavior results from the superposition of possible processing cases shown in Figure 4.7a-c on $\overline{r}$ and $\overline{AC}$.

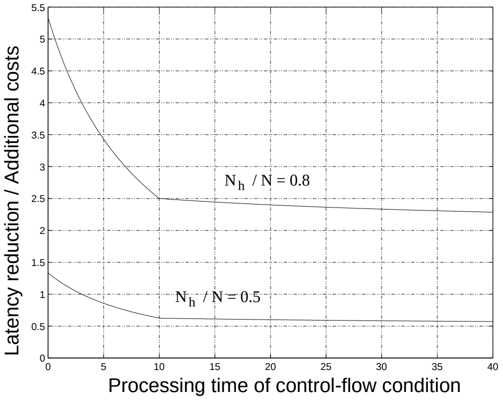


Figure 4.20: Scenario 5: Latency reduction / additional cost

The reduction of the processing delay continuously decreases with increasing values for T_{L2_C}. A local break of the graph is obtained if $T_{L2_1} + T_{L2_C} = T_{L3_1}$. i.e. if T_{L3_1} is completely processed in case of a misprediction. The additional costs and the latency reduction is depicted in Figure 4.21.

The figure shows that the additional costs possess a local maximum for $T_{L2_C} = T_{L3_1} - T_{L2_1}$. Thus, it can be concluded that processing delays for the resolution of control-flow conditions that refer to upper layers should have small processing times.

4.2.2.6 Scenario 6: Prediction model: single-level or multi-level

For the speculative processing model, the prediction delivers an information that represents the complete structure of the predicted layers, i.e. the types of the predicted layers. The verification whether the prediction of a certain layer was correct takes place in an ascending order of protocol-layer numbers. In case of a correctness, i.e. a layer was correctly

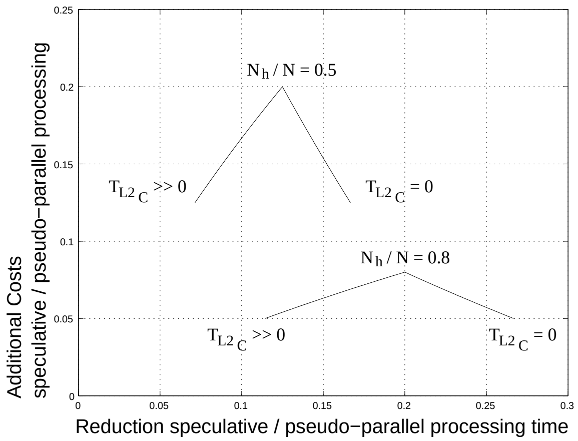


Figure 4.21: Scenario 5: $\overline{AC}, \overline{r}$ / Parameter: processing time of control-flow conditions

predicted, processing all the verified layer and the upper layers proceeds. Otherwise, processing of the verified layer and the upper layers is aborted. For the determination of upper protocol layer, two alternatives exist as shown in Figure 4.22.

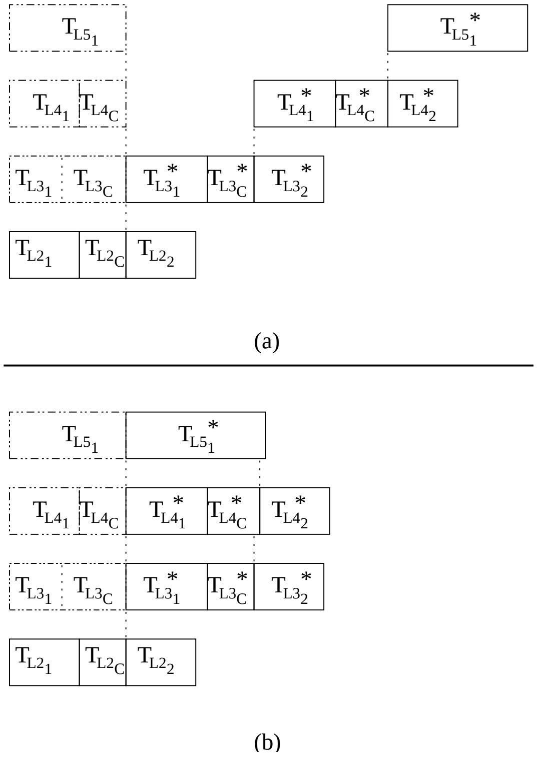
In case of a *single-level prediction* (SLP), no further prediction will be requested for the current data. The determination of upper layers as well as the further processing follows the pseudo-parallel processing model. Else, a *multi-level prediction* (MLP) predicts upper layers based on the obtained knowledge of lower layers that already have been determined. Figure 4.23 shows additional costs and the mean reduction $\overline{r}$ for $N_{ii}/N = 0.25$ and $N_{iii}/N = 0.125$.

It can be seen that MLP has an increased value for $\overline{r}$ compared to SLP. However, additional costs are likewise increased in the MLP model. If the prediction comprises layer 2 up to 5 and implements MLP, a mean reduction of 32 % is achieved.

4.3 Conclusion

The potential of the speculative processing model has been estimated through an abstract concept evaluation. For this purpose, a pseudo-parallel model has been taken as reference. Some essential observations that have been made during the analysis inspire an implementation of the speculative processing model in the next chapter.

- The prediction of multiple layers leads to additional resource costs due to probably occurring mispredictions. However, a notable delay reduction can be achieved. If a



(a)

(b)

Figure 4.22: Single-level prediction (a) and multi-level prediction (b)

multi-level prediction is applied, additional costs are required compared to a single-level prediction. However, processing delays can be significantly reduced through

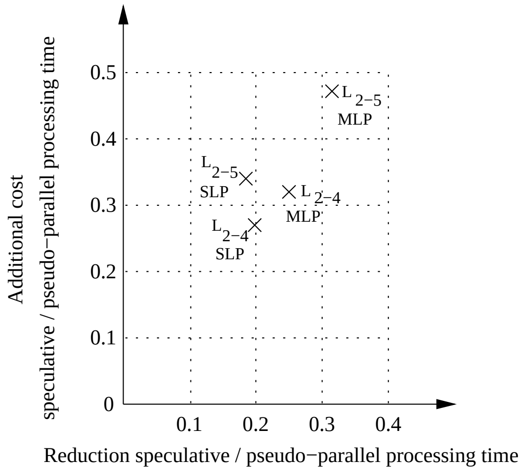


Figure 4.23: Scenario 6: $\overline{AC}, \overline{r}$ / Parameter: prediction model

multiple prediction requests.

- An essential parameter that permits a large reduction of latencies together with small cost increases is prediction accuracy. Thus, a substantial effort should be spent for realization of a prediction instance to achieve large values of the prediction hit ratio.
- The resolution times of control flow conditions have a strong impact on the mean latency reduction and the additional costs. In order to achieve a maximum value for the mean delay reduction by meeting the local maximum, the processing order of tasks and the checktask position have to be optimized.
- Maximum values for the delay reduction and minimum values for additional costs can be achieved through small processing times of checktasks.
- In case of mispredictions, additional processing delay for abortion and re-initiation of functions significantly reduces the concept's benefit. Thus, short interrupt times have to be striven for by means of appropriate mechanisms.

The observed parameters can be classified into parameters which can be optimized, e.g. by the processing flow, and parameters which are invariant due to application constraints. Examples for parameters that can be optimized are the prediction accuracy and the support of a multi-level prediction. The first parameter can be modified through an appropriate specification and implementation of a prediction algorithm. The employment of a multi-level prediction model generally depends on the presence of multiple layers. Considering networking applications, bridges are specified to comprise processing up to layer 2 and common routers up to layer 3. Routers that additionally implement protocol-processing of the transport layer analyze furthermore layer 4 as well. It has to be concluded, however, that the presence of encrypted data, which are not processed by the system, limits the value for the latency reduction. In the networking area, layer-2-tunneling and layer-3-tunneling, e.g. IPSec with tunnel-mode, are examples for encrypted packets that limit the latency reduction [105].

An optimization of the processing order of tasks and the position of a checktask commonly depend on the application, too. Considering Internet protocol-processing, tasks such as header verification or identification of the IP version are performed before others, e.g. a next-hop lookup. Consequently, a modification of the processing order is only reasonable for a limited number of tasks.

The computation time of checktasks and the additional processing time in case of mispredictions have to be optimized by a well-suited implementation of the processing model.

Chapter 5

System design

5.1 Introduction

The goal of this part is the creation of a system architecture that allows an evaluation of the proposed speculative processing concept in a networking-specific environment.

The striven result will be neither an optimized architecture implementation that allows a comparison with currently available NPs nor a development that covers backend stages of the VLSI-design flow. The design amount is restricted towards a system-level description to enable a fast application-specific evaluation of the system. This comprises a consideration of probable bottlenecks that limit system performance. Examples are task concurrencies in processor units and shared elements, e.g. busses or memory devices [47].

In order to evaluate the speculative processing model, an exemplary implementation for the system is defined. At first, the application is specified followed by the corresponding requirements. This contains the application area of the system as well as the supported protocol-stack. The functional specification is described thereafter. The functional processing flow is divided into three parts, namely input processing, protocol processing and output processing. While input and output processing are specified as partitions that are independent of the underlying processing model, protocol processing implements the speculative model and the pseudo-parallel reference model, respectively. This classification of processing parts allows a dedicated evaluation of the associated processing model. Finally, the exemplary system architecture will be explained.

5.2 Constraints of the application example

5.2.1 System environment

Routers are commonly classified into several categories according to their application, e.g. backbone routers and access or edge routers [106]. While the former is focused on fast and simple packet forwarding, the latter performs protocol-processing with an increased complexity. For example, if an edge router support CoS, a classification of a received packet has to be done when entering an ISP domain. The concept evaluation performed in the previous chapter showed that the mean latency reduction can be increased by a larger amount of predicted layers. Consequently, an application as an edge router is selected for the system environment of the processing model.

5.2.2 Supported protocols

Common protocols for edge routers are Gigabit Ethernet, Packet over SONET, ATM and, for higher protocol layers, the TCP/UDP/IP protocol suite [107] [108]. The range of supported protocols for the defined system is limited as depicted in Figure 5.1.

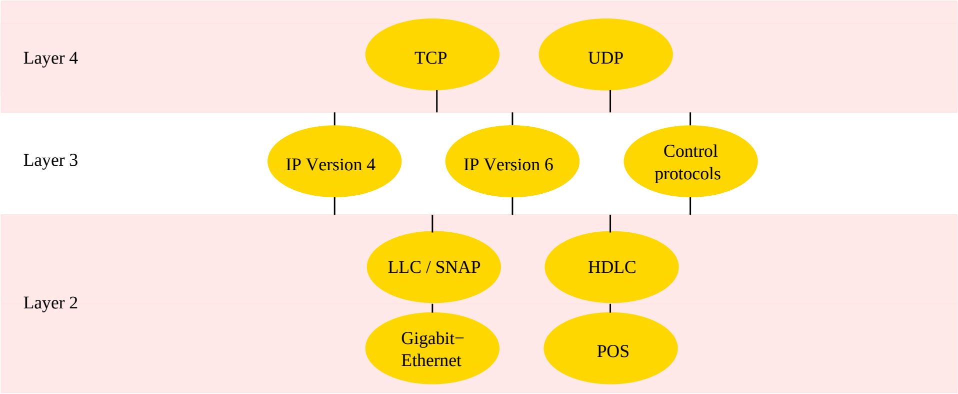


Figure 5.1: Supported protocols

The supported protocols in the figure above cover the protocol layers two up to four. The prediction algorithm, which will be introduced later, exploits the type of protocol layer 2 to differentiate the traffic which has been received and to provide a higher accuracy. The type is signalized by a POS-framer or a medium access control device before the system. The employed protocols for the physical and the link layer are Gigabit-Ethernet (*G-E*) [109] and Point-to-Point in HDLC-like framing [110]. The reason for their support is derived from their current application in edge router devices as well as initiatives which are focused on a universal protocol support for different physical and link layers (POS-PHY Level 3 (OC-48) and POS-PHY Level 4 (OC-192)) . Two examples for framer devices that support both protocol types are described in [111] and [112]. The data-link layer type is indicated to the system by the framer or MAC device. For the higher layers, the TCP/IP suite is applied, i.e. the network layer protocols are the Internet protocol versions 4 and 6. TCP and UDP are used as transport layer protocols. The employed control protocols in the figure above are the Internet Control Message Protocol (ICMP) and Open Short Path First (OSPF) .

5.2.3 Supported services

The system supports differentiated services to provide CoS. It comprises tasks for multi-field classification [113], policing and accounting [114].

5.2.4 Protocol-stack

The skeleton of the applied software protocol-stack used is adapted from the reference implementation of the Intel IXP Network Processor IXP1200 / Release 1.0 [23]. Extensions that implement differentiated services are derived from [115]. Further completions result from the Berkeley Software Distribution version 4.4 BSD-Lite) [116]. The IP version 6 code is taken from [117].

5.3 Functional specification

5.3.1 Overview

The protocol-stack processing of a packet is assigned to a processing unit, which is referred to as protocol-processing element (PPE). The functional processing flow of a PPE is depicted in Figure 5.2.

It includes all tasks that are performed after a packet has been dispatched to the PPE and before it is transfered to output transmit queues. Foregoing and succeeding tasks are performed by receive and transmit units that are external to a PPE.

Inside a PPE, *input processing* starts with the reception of a packet from the receive unit and terminates before the initiation of protocol processing. *Protocol-stack processing* comprises packet header parsing and manipulation as well as DiffServ tasks classification, policing and accounting. Forwarding is done based on the given destination address. If the network destination of the packet has been reached, i.e. if the network node represents the packet sink, packet termination is followed by local processing. In case of corrupted packet data, the packet is handled according to predefined rules, e.g. dropping or degradation of

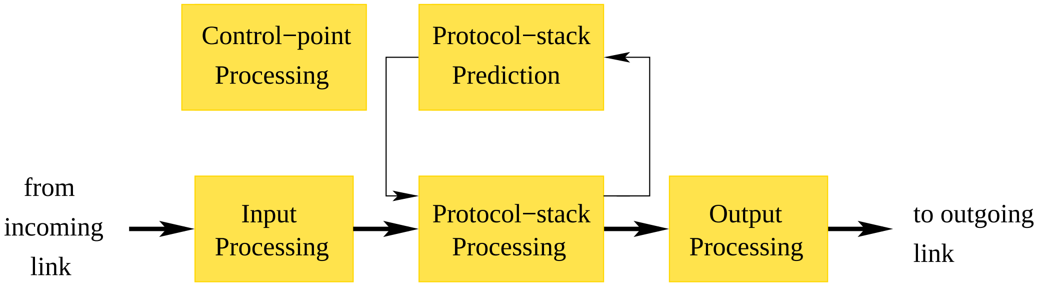


Figure 5.2: PPE processing chain

flow priority. The *output processing* block merges PCI and PDU and delivers packets to output queues. In addition, a marker that indicates occurred mispredictions of the packet is added. *Protocol-stack prediction* provides the system with a label that characterizes the expected protocol-stack of future packets and determines the flow of protocol-stack processing.

In order to specify processes that take place in the processing chain, a conditional process graph (CPG) is used. A CPG is a directed, acyclic, polar graph and allows a convenient system representation by an abstract model [118]. The graph enables a description of the process schedule in the context of both, control and data dependencies.

5.3.2 Input processing

The input processing flow is illustrated in Figure 5.3.

When a packet is received by a PPE, it is completely transfered to the input buffer. The assigned protocol type of the data-link layer is signalized to the control-point PU (CP) of the PPE. For the supported protocols, the lengths of the transferred PCIs to the NL-PUs for layer 2, 3 and 4 are derived from the specification of the supported protocols. These PCIs are sequentially sent to NL-PU 2, 3, and 4. Finally, the complete packet is moved into DRAM. After the CP has noticed completion of data transfer, protocol-stack processing is initiated.

If input processing and protocol-processing are interleaved, i.e. protocol-processing in a NL-PU starts immediately after the transfer of the corresponding layer PCI, a performance increase for the PPE will be expected. A similar impact on the speculative as well as on a reference system that differs merely in the underlying processing model could be assumed. This modification of the input processing flow is not realized in this part.

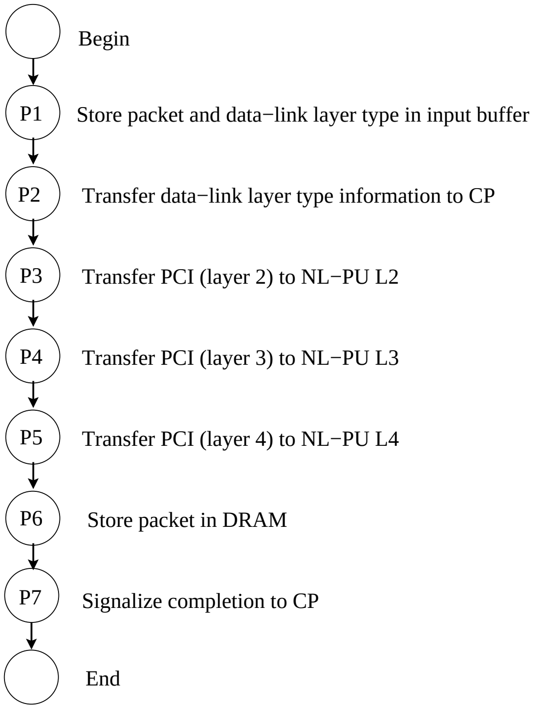

Begin
P1
Store packet and data−link layer type in input buffer
P2
Transfer data−link layer type information to CP
P3
Transfer PCI (layer 2) to NL−PU L2
P4
Transfer PCI (layer 3) to NL−PU L3
P5
Transfer PCI (layer 4) to NL−PU L4
P6
Store packet in DRAM
P7
Signalize completion to CP
End

Figure 5.3: Input processing flow

5.3.3 Protocol-stack processing

The flow of protocol-stack processing is illustrated in Figure 5.4.

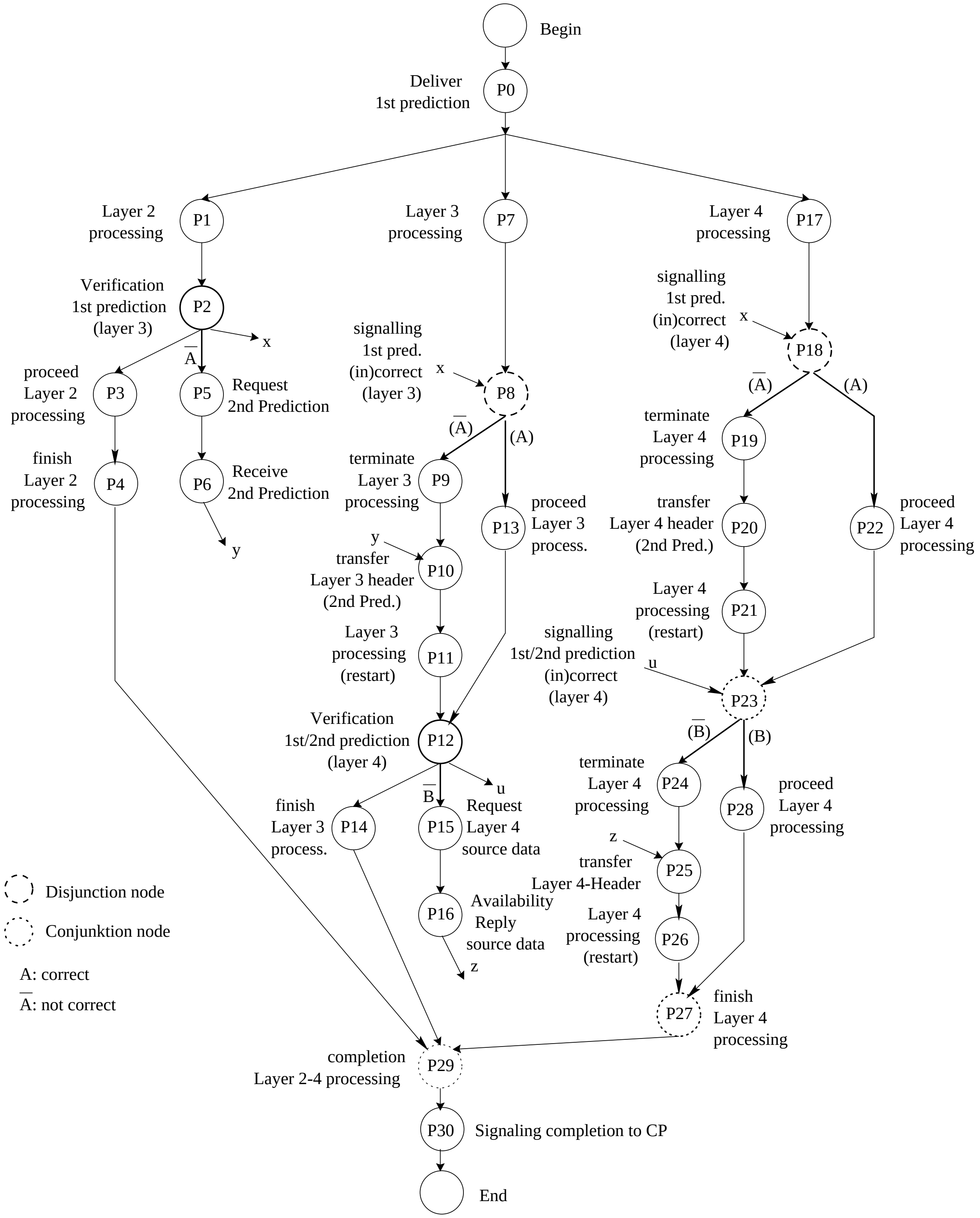


Figure 5.4: Protocol-stack processing flow

CPG disjunction nodes represent nodes that possess conditions at their edges. They are highlighted through dashed lines. Depending on the resolution of the condition, one of the alternative paths is taken by the process flow. In CPG conjunction nodes, multiple paths are joined, which is shown by dotted lines.

The concurrency of the network layers two up to four in the figure is illustrated by a graphical arrangement in columns. After reception of the first prediction, protocol processing is invoked in parallel. Process $P2$ resolves the data dependency concerning layer three. The corresponding packet header field is therefore parsed from layer two. The result of the prediction is a value that represents the protocol-stack of a packet and allows to derive the contained types of the several protocol-layers for the packet. The verification of the prediction is finally done through a comparison of the actually present protocol layer types of the packet and of the predicted types. Layer-two processing is completed.

In case of a misprediction, a second prediction is requested ($P5$). The signaling of a performed verification, indicated by x, is sent to layer three and four. Depending on the obtained result, one of two paths at the output of node $P8$ is taken. In case of a misprediction, layer-three processing is terminated.

Layer-three header data might be no longer available due to overwriting of register values. In that case, header data has to be transferred ($P10$) from DRAM and processing can be revoked ($P11$).

In case of a prediction hit (A), layer three processing is continued. The predicted protocol type for layer 4 has to be verified as well ($P12$). Finally, layer three processing ends in $P14$.

In case of a misprediction concerning layer 4, $P15$ requests overwritten source header data of layer four from the memory.

Layer four processing takes place in a similar way. In $P18$, the signaling is done for a correct or wrong first prediction and in $P23$ for the second prediction. Protocol-stack processing ends in $P29$ when all layer processing is completed and the control-point PU gets informed ($P30$).

The processing tasks in the figure cover the processing chain in Figure 2.1 without packet reception and transmission as well as queuing and scheduling.

5.3.4 Output processing

After completion of protocol-processing, the initiation of output processing is invoked by the control-point. The flow is shown in Figure 5.5.

The CP triggers NL-PU 2, 3 and 4 to send corresponding parts of the protocol layer header to the output buffer. As a result of protocol processing, required control information concerning PCI boundaries is gathered by the CP. Hereafter, the PDU is fetched from the memory and is kept in the output buffer. Furthermore, additional data, which is required

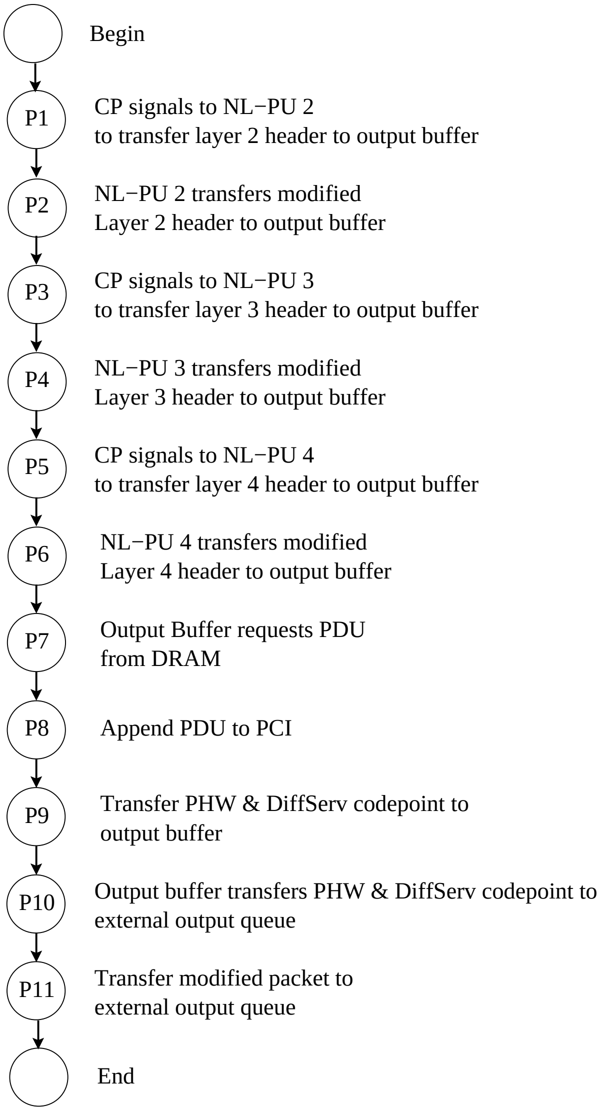

Begin
P1
CP signals to NL−PU 2
to transfer layer 2 header to output buffer
P2
NL−PU 2 transfers modified
Layer 2 header to output buffer
P3
CP signals to NL−PU 3
to transfer layer 3 header to output buffer
P4
NL−PU 3 transfers modified
Layer 3 header to output buffer
P5
CP signals to NL−PU 4
to transfer layer 4 header to output buffer
P6
NL−PU 4 transfers modified
Layer 4 header to output buffer
P7
Output Buffer requests PDU
from DRAM
P8
Append PDU to PCI
P9
Transfer PHW & DiffServ codepoint to
output buffer
P10
Output buffer transfers PHW & DiffServ codepoint to
external output queue
P11
Transfer modified packet to
external output queue
End

Figure 5.5: Output processing flow

for subsequent flow-based and prediction-miss-based scheduling, is sent from the CP to the output buffer. Finally, packet data and additional information are transferred to the external output queue.

5.3.5 Control-point processing

Control-point processing executes tasks that refer to the control-plane. For example, routing protocols which exchange routing information of the network are transferred to each network node. This processing is less time-critical. Consequently, it is done asynchronously to data forwarding.

The synchronization of protocol-layer processing, the administration of PPE resources, error detection and the synchronization with external system resources is furthermore part of control-point processing.

5.3.6 Protocol-stack prediction

The system provides a lookup table to access the prediction. The determination of the prediction value, i.e. the algorithm still has to be explained.

The goal of the prediction algorithm is to provide a label that identifies the protocol-stack of the packet that will be received next. The applied algorithm for protocol-stack prediction relies on MFU, i.e. prediction output values refer to the protocol-stacks which were most-frequently received before. In case of equal values for reception frequencies, the least-recently used (*LRU*) protocol-stack is predicted.

The algorithm comprises two stages. First, only the data-link layer is known and a first prediction label is requested. In case of prediction failure, the algorithm provides a second prediction that considers the network layer type which has been found out in the meantime. Thus, with respect to the supported protocols, the algorithm delivers as output one of the following MFU entries listed in Table 5.1.

It has to be mentioned that the entries in the table above are exclusive. If the network layer of MFU_0 is IPv4, for example, MFU_2 is the protocol-stack with the second highest frequency, but with a different transport layer protocol.

In order to determine the most-frequently used entries, a traditional sorting algorithm could be applied to derive a frequency order for a set of data. Common algorithms are heap sort, bubble sort, insertion sort, quicksort (partition sort), etc. [119]. Sorting methods for N data elements require between $\approx log(N) \cdot N$ and $\approx N^2$ compare instructions [120]. Considering the two stages of the prediction scheme and two protocol types for layer 2 and 3, in total 6 exclusive MFU values have to be determined. In order to perform prediction updates at line speed, the drawbacks mentioned above force an alternative approach. Instead, a table

MFU_0	most frequently used protocol-stack with Gigabit-Ethernet as data-link layer
MFU_1	most frequently used protocol-stack with POS-HDLC as data-link layer
MFU_2	most frequently used protocol-stack with Gigabit-Ethernet as data-link layer and IP version 4 as network layer, but not MFU_0
MFU_3	most frequently used protocol-stack with Gigabit-Ethernet as data-link layer and IP version 6 as network layer, but not MFU_0
MFU_4	most frequently used protocol-stack with POS-HDLC as data-link layer and IP version 4 as network layer, but not MFU_1
MFU_5	most frequently used protocol-stack with POS-HDLC as data-link layer and IP version 6 as network layer, but not MFU_1

Table 5.1: Most-frequently used entries

scheme is applied for the dynamic calculation of prediction values that requires a smaller number of compare instructions. The algorithm is shown in Figure 5.6.

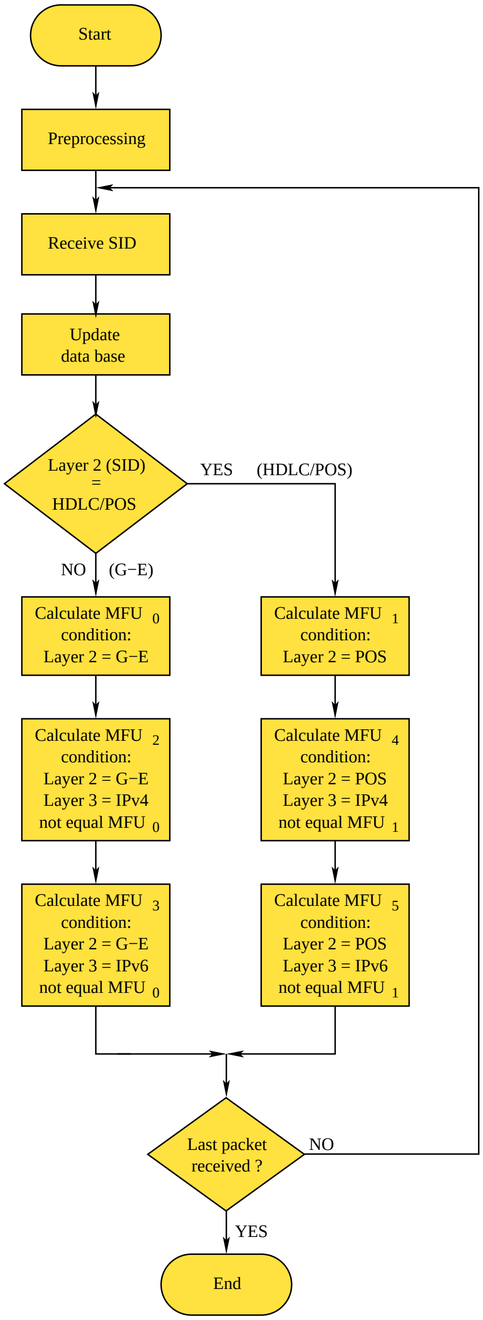

Start
Preprocessing
Receive SID
Update data base
Layer 2 (SID) = HDLC/POS
YES (HDLC/POS)
NO (G−E)
Calculate MFU 0 condition: Layer 2 = G−E
Calculate MFU 1 condition: Layer 2 = POS
Calculate MFU 2 condition: Layer 2 = G−E Layer 3 = IPv4 not equal MFU 0
Calculate MFU 4 condition: Layer 2 = POS Layer 3 = IPv4 not equal MFU 1
Calculate MFU 3 condition: Layer 2 = G−E Layer 3 = IPv6 not equal MFU 0
Calculate MFU 5 condition: Layer 2 = POS Layer 3 = IPv6 not equal MFU 1
Last packet received ?
NO
YES
End

Figure 5.6: MFU computation flow

Step 1: During preprocessing, initialization of the special register file in the prediction PU can be done if a traffic profile is available. Subsequent to that, register copies are sent to the PPE-local DHTs.

Step 2: After protocol-processing of a packet has been completed, the corresponding stack identifier (SID) is transferred from the PPE to the prediction PU.

Step 3: The database which contains the reception frequencies of the supported protocol-stacks is updated. According to the system specification, in total 16 different protocol-stacks are supported according to Table 5.2.

Step 4: Depending on the data-link layer type associated with the current SID, the most-frequently used stacks might have changed. Thus, in case of Gigabit-Ethernet (G-E), MFU_0 has to be verified. Else, i.e. HDLC/POS, MFU_1 has to be checked. If the stack that refers to the current SID occurred more often, the previous MFU entry will be replaced.

Step 5: If one of the previous entries MFU_0 or MFU_1 has been replaced, its frequency might be still higher than the corresponding entry of $MFU_2 ... MFU_5$ entry that has the same protocol types for layer 2 and 3. Consequently, the corresponding entry $MFU_2 ... MFU_5$ is replaced by the previous MFU_0 or MFU_1. If the protocol-stack of the current packet has not been received more often than the corresponding entry of MFU_0 and MFU_1 respectively, its frequency might be nevertheless higher than the frequency of the corresponding entry of $MFU_2 ... MFU_5$. In this case, only one replacement for the corresponding entry of $MFU_2 ... MFU_5$ takes place.

Step 6: If the system is still in operation, the following packet invokes a database update and MFU calculation again.

Table 5.2 shows the assignment of protocol layers to a value for the stack identifier (SID).

Then, the algorithm selects the first two MFU according to:

$$MFU_0 = i \quad with \quad n_i = max\{w_0 \cdot n_0; w_2 \cdot n_2; w_4 \cdot n_4; ...; w_{14} \cdot n_{14}\} \tag{5.1}$$
$$MFU_1 = i \quad with \quad n_i = max\{w_1 \cdot n_1; w_3 \cdot n_3; w_5 \cdot n_5; ...; w_{15} \cdot n_{15}\} \tag{5.2}$$

with the number of received packets n_i of protocol-stack (SID) i. Delay-sensitive applications require short-end-to-end latencies. In order to prioritize these protocol types, a variable weight factor w_i is defined. While its default value is $w_i = 1$, $w_i > 1$ is used to assign priority to stack i.

The remaining MFU entries are calculated according to

SID	Layer 4	Layer 3	Layer 2
0	TCP	IPv4	G-E
1	TCP	IPv4	POS
2	TCP	IPv6	G-E
3	TCP	IPv6	POS
4	UDP	IPv4	G-E
5	UDP	IPv4	POS
6	UDP	IPv6	G-E
7	UDP	IPv6	POS
8	ICMP(4)	IPv4	G-E
9	ICMP(4)	IPv4	POS
10	ICMPv6	IPv6	G-E
11	ICMPv6	IPv6	POS
12	OSPF	IPv4	G-E
13	OSPF	IPv4	POS
14	OSPF	IPv6	G-E
15	OSPF	IPv6	POS

Table 5.2: Protocol-stack identifier

$$\begin{aligned} MFU_2 = i \quad with \quad & n_i = max\{(w_0 \cdot n_0; w_4 \cdot n_4; w_8 \cdot n_8; w_{12} \cdot n_{12}) \\ & \setminus max\{w_0 \cdot n_0; w_2 \cdot n_2; w_4 \cdot n_4; ...; w_{14} \cdot n_{14}\}\} \end{aligned} \tag{5.3}$$

$$\begin{aligned} MFU_3 = i \quad with \quad & n_i = max\{(w_2 \cdot n_2; w_6 \cdot n_6; w_{10} \cdot n_{10}; w_{14} \cdot n_{14}) \\ & \setminus max\{w_0 \cdot n_0; w_2 \cdot n_2; w_4 \cdot n_4; ...; w_{14} \cdot n_{14}\}\} \end{aligned} \tag{5.4}$$

$$\begin{aligned} MFU_4 = i \quad with \quad & n_i = max\{(w_1 \cdot n_1; w_5 \cdot n_5; w_9 \cdot n_9; w_{13} \cdot n_{13}) \\ & \setminus max\{w_1 \cdot n_1; w_3 \cdot n_3; w_5 \cdot n_5; ...; w_{15} \cdot n_{15}\}\} \end{aligned} \tag{5.5}$$

$$\begin{aligned} MFU_5 = i \quad with \quad & n_i = max\{(w_3 \cdot n_3; w_7 \cdot n_7; w_{11} \cdot n_{11}; w_{15} \cdot n_{15}) \\ & \setminus max\{w_1 \cdot n_1; w_3 \cdot n_3; w_5 \cdot n_5; ...; w_{15} \cdot n_{15}\}\} \end{aligned} \tag{5.6}$$

The exclusivity of the entries MFU_0 and MFU_1 and $MFU_2 ... MFU_5$ results from the applied replacement strategy of the MFU entries. Its purpose is to avoid a complex calculation effort for sorting the reception frequencies of the supported protocol-stack after each processing of each packet.

The flow of protocol-stack prediction as a part of protocol-processing, i.e. the two prediction stages and the prediction table update, are highlighted on the left by dark rectangles in Figure 5.7. On the right, the corresponding state of the knowledge for individual protocol-layers of a received packet is shown. While bright boxes represent unknown layers, dark rectangles signify already analyzed and consequently known layers.

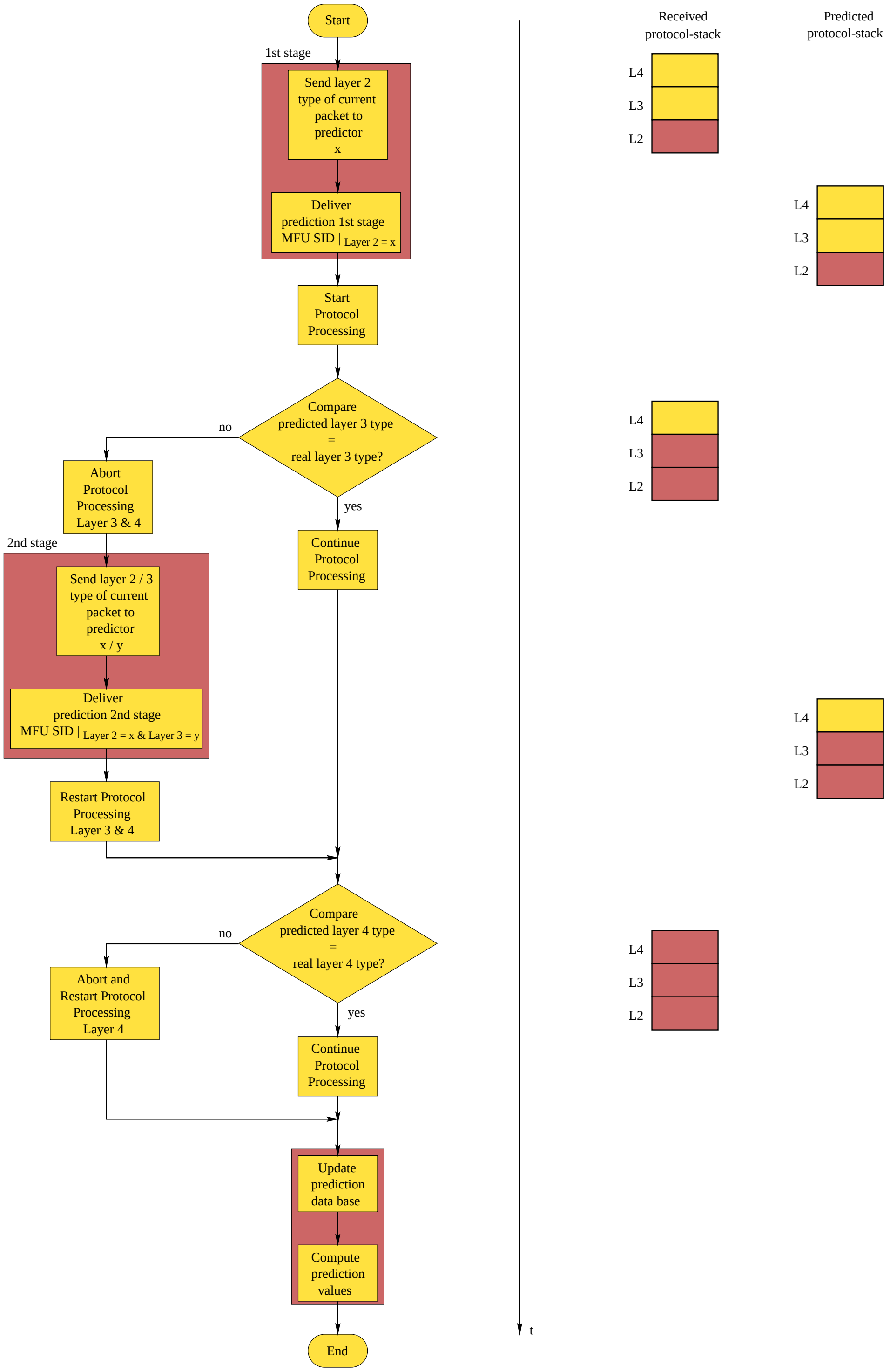

Start
1st stage
Send layer 2 type of current packet to predictor x
Deliver prediction 1st stage MFU SID | Layer 2 = x
Start Protocol Processing
Compare predicted layer 3 type = real layer 3 type?
no
yes
Abort Protocol Processing Layer 3 & 4
2nd stage
Send layer 2 / 3 type of current packet to predictor x / y
Deliver prediction 2nd stage MFU SID | Layer 2 = x & Layer 3 = y
Restart Protocol Processing Layer 3 & 4
Continue Protocol Processing
Compare predicted layer 4 type = real layer 4 type?
no
yes
Abort and Restart Protocol Processing Layer 4
Continue Protocol Processing
Update prediction data base
Compute prediction values
End
t
Received protocol-stack
Predicted protocol-stack
L4
L3
L2

Figure 5.7: Components of protocol-stack prediction

The prediction values are kept in a set of special registers. Beside the SID, which represents the predicted protocol-stack, each register pair contains a register where the reception frequency of the corresponding stack is stored. The six pairs on the left refer to MFU_i with $i \in \{0..5\}$ which correspond to Table 5.1.

The special register file is depicted in Figure 5.8.

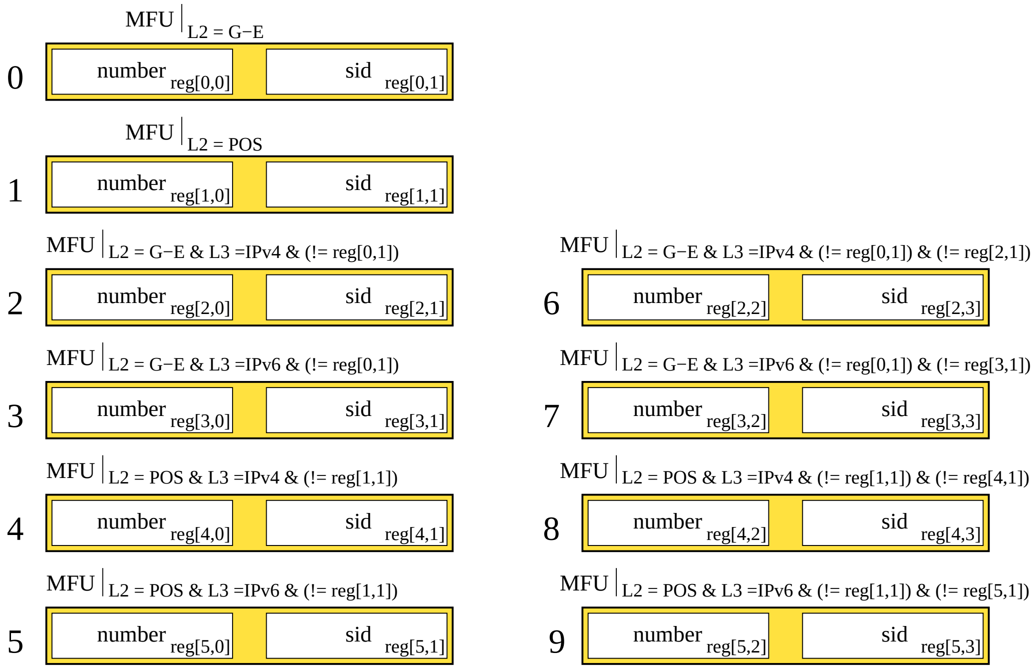

MFU |L2 = G−E
0 number reg[0,0] sid reg[0,1]
MFU |L2 = POS
1 number reg[1,0] sid reg[1,1]
MFU |L2 = G−E & L3 =IPv4 & (!= reg[0,1])
2 number reg[2,0] sid reg[2,1]
MFU |L2 = G−E & L3 =IPv6 & (!= reg[0,1])
3 number reg[3,0] sid reg[3,1]
MFU |L2 = POS & L3 =IPv4 & (!= reg[1,1])
4 number reg[4,0] sid reg[4,1]
MFU |L2 = POS & L3 =IPv6 & (!= reg[1,1])
5 number reg[5,0] sid reg[5,1]
MFU |L2 = G−E & L3 =IPv4 & (!= reg[0,1]) & (!= reg[2,1])
6 number reg[2,2] sid reg[2,3]
MFU |L2 = G−E & L3 =IPv6 & (!= reg[0,1]) & (!= reg[3,1])
7 number reg[3,2] sid reg[3,3]
MFU |L2 = POS & L3 =IPv4 & (!= reg[1,1]) & (!= reg[4,1])
8 number reg[4,2] sid reg[4,3]
MFU |L2 = POS & L3 =IPv6 & (!= reg[1,1]) & (!= reg[5,1])
9 number reg[5,2] sid reg[5,3]

Figure 5.8: Special register file

The four additional register pairs, i.e. MFU_i with $i \in \{6..9\}$, are required for the applied replacement strategy of MFU entries. If a received packet causes that another protocol-stack becomes the MFU_i with $i = 0or1$, the current MFU possesses from now on the second highest number of reception events. Instead of dropping this information, it will be kept in the corresponding MFU_i with $i \in \{2..9\}$. Thus, the special register file gathers not only the numbers of reception events for the MFUs, but also the protocol-stacks with the second highest number of reception events. If another protocol-stack becomes MFU, a transfer of MFU register values from MFU_i with $i = 0or1$ to one of the MFU_i with $i \in \{2..5\}$ takes place. The necessity of the register pairs MFU_i with $i \in \{6..9\}$ results from preserving completeness of MFU with the highest and the second highest number of packet reception events. It can be explained by the following example:

At first, the following assumptions are exemplarily made: $MFU_0 := 2$, $MFU_2 := 4$, $MFU_6 := 8$ and $n_2 := n_4$. If the current stack corresponds to SID 4, then the entry of MFU_0 is replaced by MFU_2. However the previous SID in MFU_0 is not moved to MFU_2

but to MFU_3. In order to preserve completeness of the left column, the SID of MFU_6 is moved to the left column, i.e. $MFU_2 = 8$.

In order to exemplarily illustrate right shifts of registers values, it is assumed that $MFU_0 := 0$, $MFU_2 := 4$, $MFU_6 := 8$ and $n_0 := n_4 := n_8 := n_{12}$. Now, a right shift occurs, if the protocol stack of the current packet refers to SID 12. In this case, one obtains $MFU_0 := 12$, $MFU_2 := 0$ and $MFU_6 := 4$.

The demand for a dynamic adaptation scheme of the prediction has been mentioned in chapter 3. Since the time stamps of the received packets are not kept due to an increased effort, an alternative method for the selection is used. The scheme that determines a limited number of prediction input data is shown in Figure 5.9.

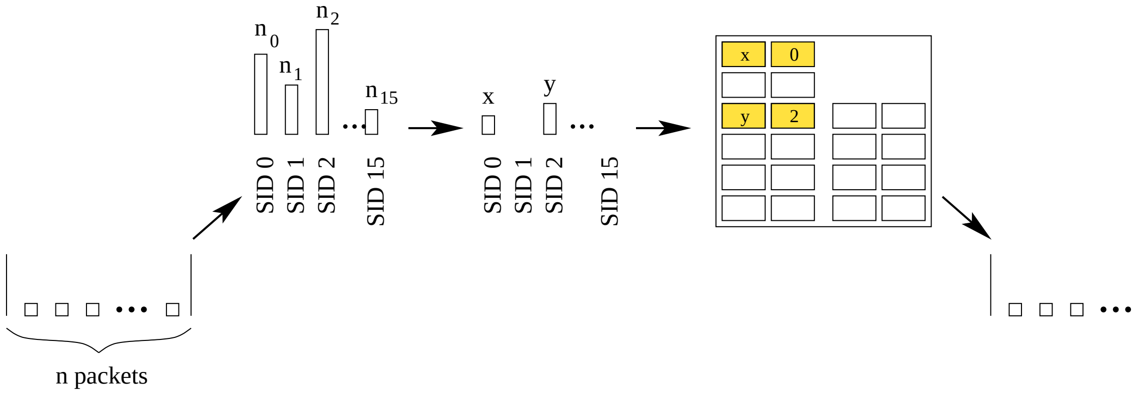


Figure 5.9: Prediction table reset facility

Frequencies of received packets are continuously taken to compute the prediction values. After the reception of n packets, the periodical method determines two protocol-stacks that have been most-frequently used during this period. The frequency numbers of all remaining SIDs are reset to zero, the frequency for the most-frequently used stack is set to x and the second is set y. This procedure allows to continuously provide a prediction value. Based on these initialization values, the calculation of prediction values proceeds. The scheme enables a transition to changed prediction values based on a modified traffic statistic. The cyclic reset avoids furthermore overflows in case of a register implementations of the packet reception frequencies.

5.4 System architecture

Research of communication systems has demonstrated that a performance increase can be achieved though an exploitation of data parallelism [121] [122] [123] [124]. Furthermore, recent studies concerning network processor architectures, which target on data rates of

OC-48 and higher, have confirmed the demand for a parallelized data path [75] [20] [125] [126] [127]. Thus, the starting point for the defined system architecture are identical processing units that implement multiprocessing.

5.4.1 Overview

The system model is illustrated as a queuing system that is composed of two stages as shown in Figure 5.10.

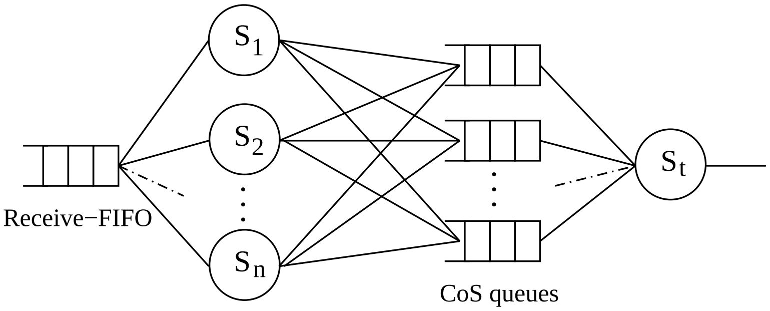


Figure 5.10: System model

Packets which are received by an external device are transferred to a FIFO buffer. If one of the servers S_i with $i \in \{1,...,n\}$ is idle, the first packet in the FIFO is dispatched. It is assumed that one server possesses the capability to completely process all protocol-stack tasks. After completion, the modified packet is stored in the queue that corresponds to its CoS label. The DiffServ queues have different service priorities, which are considered by the subsequent transmission server S_t. The corresponding architecture model of the network processor is depicted in Figure 5.11.

It is realized by multiple identical packet-processing elements (PPE) that correspond to the server in the previous figure. The prediction processing unit calculates and provides the prediction values for the PPEs. One global control-point processing unit is in charge of system administration. An external global SRAM contains information that is required for routing and differentiated services support. In order to achieve accelerated lookup times through avoidance of accesses on shared resources, an SRAM controller is used to enable copying memory content to the PPE local SRAM. Packets which are forwarded from the framer to the system are dispatched to an idle PPE.

A hybrid architecture model is chosen to reduce the synchronization overhead during protocol processing: the parallel PPEs implement message parallelism while network-layer parallelism is given within the PPEs. Each of them possesses the capability to store one packet and to handle one packet per time. If protocol processing of a packet is completed,

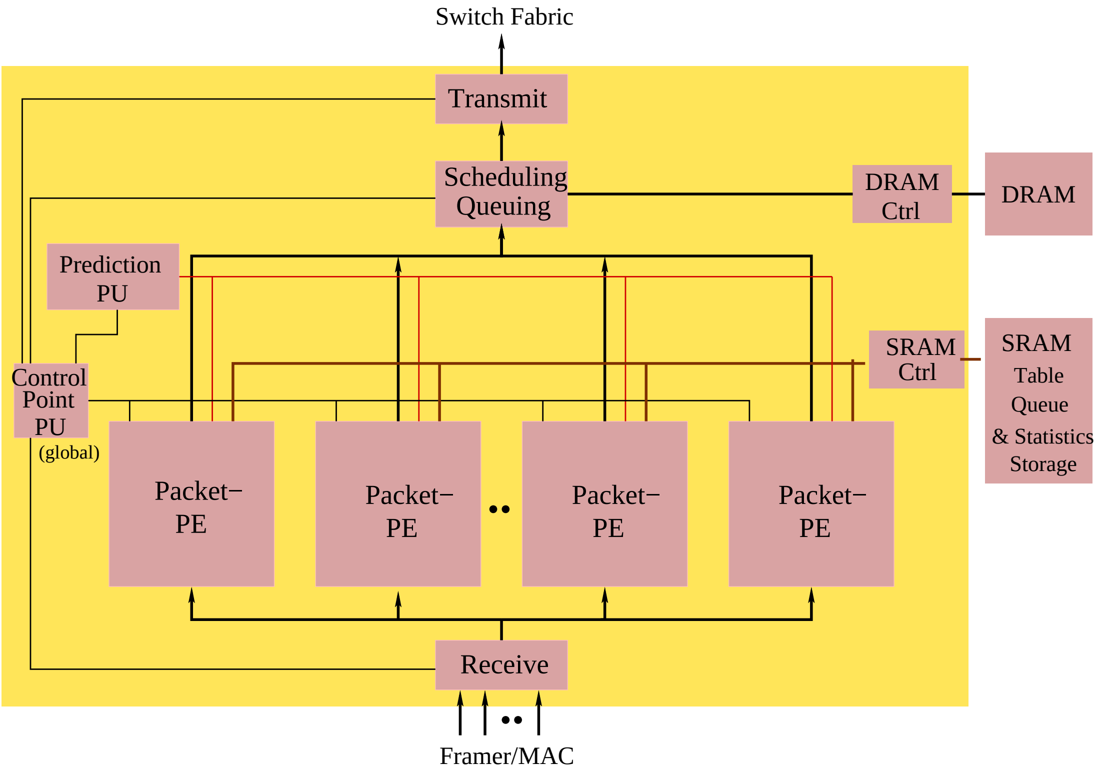


Figure 5.11: System architecture

the packet is transferred to an output queue that corresponds to its per hop behavior. According to the output link rate of the system and the queuing discipline, packets are served and finally transmitted to subsequent devices.

5.4.2 Packet-processing element architecture

All functionalities that refer to packet processing besides output queuing are processed by a PPE. The internal structure of a PPE is illustrated in Figure 5.12.

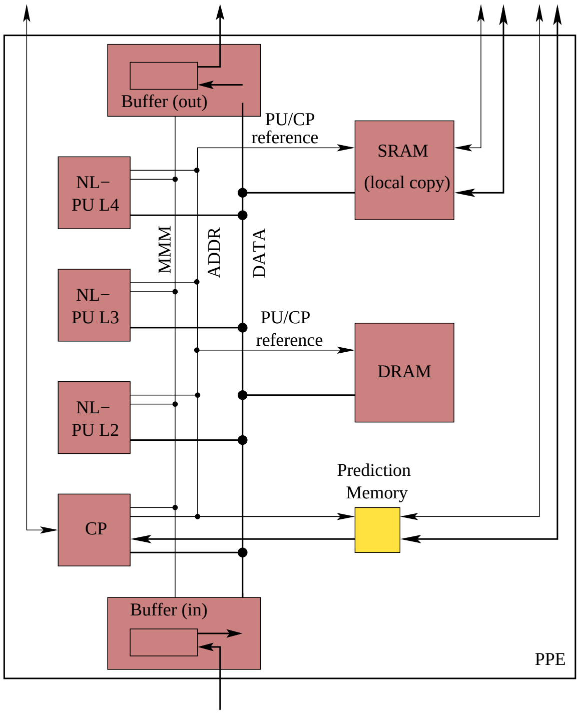


Figure 5.12: Packet Processing Element

A PPE is built up of three network layer-processing units (NL-PU), one control-point processing unit (CP) and three memories. NL-PUs and the CP are realized as embedded processors. Separate input and output interfaces possess the capability to buffer a complete packet. The master-master message bus (MMM) is used for control signaling between master units. Data exchange between modules is done through the address (ADDR) and data bus (DATA). The width of the data bus (DATA) is 32 bit.

An arbiter, which is not depicted, gives access to the shared busses. The access is granted according to a predefined order. The highest priority is given to the CP, followed by the NW-PUs in ascending order. The lowest access priority is assigned to the input and output buffers.

Inside each PPE, an embedded RAM for packet buffering is implemented. Similar NP realizations implement packet buffers with a size of multiple KByte, e.g. 12 KByte embedded data memory of a channel processor for the C-Port C-5 NP [128] and 128 KByte embedded SRAM in an IBM Rainier NP [22]. For this exemplary implementation, the packet RAM of a PPE is defined to 96 Bytes. This size corresponds to the length of packets during the succeeding system evaluation.

The calculated prediction values are transferred from the prediction PU to the prediction memory in the PPEs. The CP of a PPE can access updated prediction values hereafter.

Considering the assignment of functions, tasks that follow frame processing, which is done externally, and refer to layer two are performed by the NL-PU L2. While NL-PU L3 covers network layer processing, DiffServ functions are assigned to NL-PU L4.

5.4.3 Network-layer processing unit and control-point PU

Processing tasks of a layer are handled by a network-layer processing unit. The internal architecture of a NL-PU is illustrated in Figure 5.13.

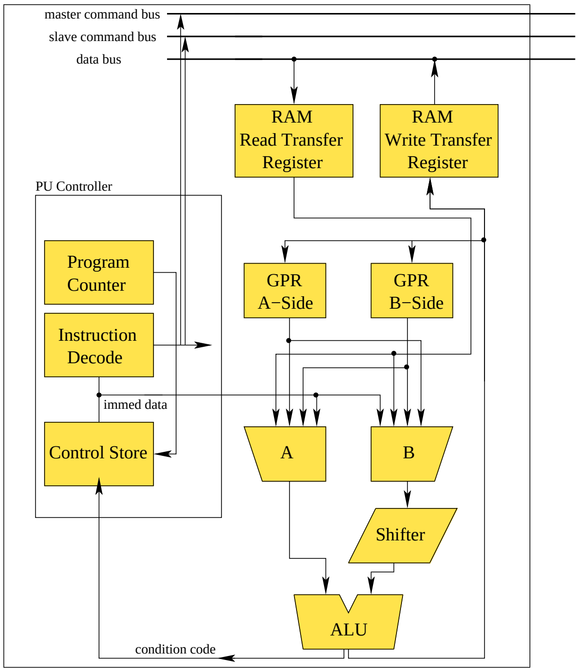


Figure 5.13: Network-layer processing unit

The three NL-PU have identical architectures that are dedicated to protocol processing. The architecture model and the RISC instruction set are derived from the Intel IXP 1200 network processor [75] [129]. It should be noted that multithreading, i.e. the capability for context switching is not implemented. Instead, a model that implements single-scalar processing is used. While general purpose registers (GPR) can be used to store data, RAM transfer registers support data transfer to and from DRAM as well as SRAM. Generally, the internal width for registers and busses is 32 bit. A NW-PU contains an ALU and a shifter which are capable of performing an ALU and shift operation in one processor cycle. The ALU can perform addition, subtraction, and logical operations as well as generation of condition codes based on these operations.

The control-point PU is responsible for PPE control, e.g. synchronization of NL-PUs and external devices as well as error detection. The architecture of a CP is defined identical to the NL-PUs.

5.4.4 Prediction unit

Protocol-stack frequencies of received packet, which are required for the compute of prediction values, are accumulated in the register file of the prediction unit. While one part of the register file is used to monitor frequency events for the protocol-stack of received packets, the other part can be exploited for working register purposes. The data-path of the prediction-processing unit is depicted in Figure 5.14.

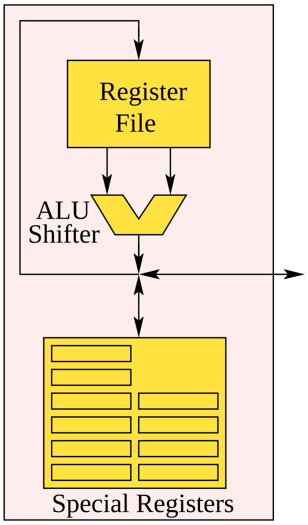


Figure 5.14: Prediction Processing Unit

After completion of protocol-processing in a PPE, the information that characterizes the protocol-stack of the current packet is transferred to the corresponding register of the register file. The content of this register is incremented by one and prediction computation is initiated. An ALU and a shifter as well as a register file are required for the calculation of prediction values. Multiple values that represent the predicted protocol-stack of a future received packet are the results. These values are stored in a set of special registers. Its content is mirrored to prediction memories within the PPEs, so-called Decode History Tables (DHT). Subsequently, a PPE can locally request a prediction label for packet processing from this DHT. The special register file is explained together with the algorithm in detail in chapter 5.3.6.

Figure 5.15 illustrates the access methodology on a DHT within a PPE. The six lines of the DHT contain the copied SIDs. The origin values are stored in the registers as depicted in the left column of Figure 5.8.

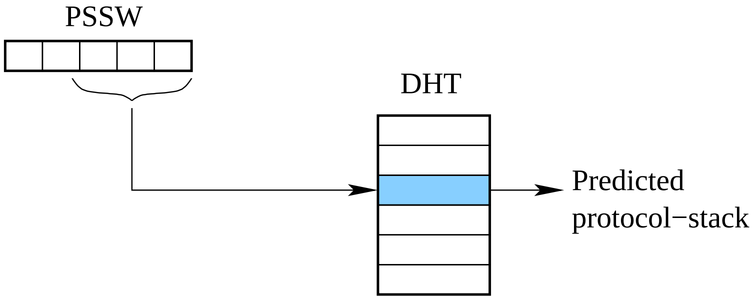


Figure 5.15: Multi-modal protocol-stack prediction

The control-point processing unit in a PPE keeps a protocol-stack status word (PSSW) to address the corresponding prediction value in a DHT.

A PSSW contains information about the current processing state of a PPE analogous to the program-status word in microprocessor, which keeps the instruction address, CPU condition codes and further state informations [130]. The format of the PSSW is illustrated in Figure 5.16.

Layer4 type	Layer4 flag	Layer3 type	Layer3 flag	Layer2 type

Figure 5.16: Protocol-stack status word (PSSW)

Flag fields indicate if particular protocol-layers have been already derived from the packet header. The corresponding protocol type is kept in the Layer-i type field. An alternative realization for a prediction access with a bit mask can be done as well.

Chapter 6

System evaluation

6.1 Introduction

The previously described speculative system architecture is now evaluated using a two-step approach. First, the evaluation of the prediction algorithm is performed, which is implemented in the prediction processing unit. For this purpose, an appropriate test methodology is defined. The applied protocol distributions are derived from current network traffic statistics.

The obtained results for the accuracy of the prediction algorithm will be exploited in the second part, i.e. for the simulation of a packet-processing element. Packet processing is decoupled through separate busses and packet memories of the PPEs. Based on these findings, the performance values for the speculative processing can be determined by individual simulations of PPEs. The PPE has been modeled with SystemC to refine embedded PUs and components, e.g. memories and busses, towards a well-suited design-level which enables a high-level performance evaluation. The system-level description language furthermore enables an annotation of timing data.

It is worth mentioning that the dynamic behavior of the prediction algorithm is evaluated within the first part. The test methodology of the PPE simulation uses a prediction table which is configured at simulation start and remains constant during operation, i.e. a static prediction table. However, a limitation of the obtained results in the second evaluation part is not given due to an application of different simulation scenarios.

6.2 Prediction PU

6.2.1 Test methodology and simulation traffic

As previously shown in section 4.2.2, a significant reduction of protocol processing delays in conjunction with low additional processing costs is strictly related to high quantities for the prediction accuracy. Furthermore, if protocol distributions vary over time, the demand for a fast adaptation of protocol-stack prediction arises. Thus, the applied test scenarios have to focus on both aspects. The simulation environment of the prediction PU is shown in Figure 6.1.

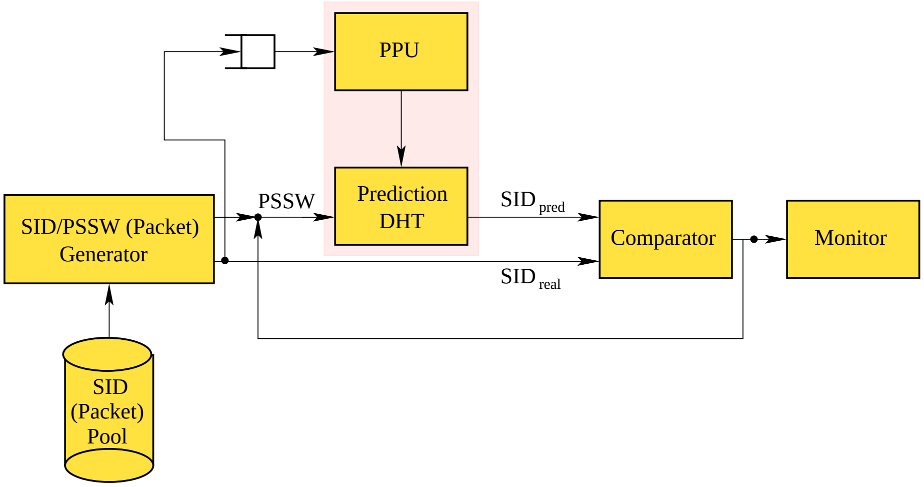


Figure 6.1: Prediction PU evaluation test bench

The simulation compares actual protocol-stack identifiers SID_{real} with predicted identifiers SID_{pred}. The simulation stimuli data is delivered by a “packet” generator. Packet data means protocol-stack identifiers that correspond to network packets. The protocol-stack status word is derived from SID_{real} and is used to request a prediction value SID_{pred}. A comparator checks the equality of both, SID_{real} and SID_{pred}. In case of a misprediction, the PSSW is updated with the actual type of protocol layer three and accesses for a second time the prediction table. A monitor device finally collects the results for both prediction stages.

At the end of protocol-processing, the system has determined the actual stack-identifier. In order to consider the packet for a next prediction computation, the simulation input SID_{real} is sent to the prediction processing unit (PPU). The processing delay is expressed by a queue with a single entry. The prediction table is consequently updated by the PPU each packet generation cycle.

Two types of simulation traffic are used during this evaluation:

- Deterministic traffic
- Random traffic

The deterministic traffic is used to analyze particular compositions of simulation inputs. Random traffic allows to stimulate the system with input data that have a predefined protocol distribution, but possess a randomized order of packet transmission. The applied protocol distributions are derived from measurements in public infrastructures:

- Traffic statistics of 1997, provided by Cooperative Association for Internet Data Analysis, University of California's San Diego Supercomputer Center (CAIDA) and the Measurement and Analysis Team (MOAT) at the National Laboratory for Applied Network Research (NLANR) [100] [131]. The gathered data were taken from the commercial Internet backbone of MCI Worldcom, and have already been used for simulation of network processor hardware [132].
- Traffic statistics of 2002, provided by the IP monitoring project of Sprint Cooperation[1]. Monitoring of data has been deployed in the Sprint E-Solutions backbone network.

Both traffic distributions show unequal characteristics. In order to find out weaknesses concerning the prediction accuracy, the simulation stimuli are extended by traffic that possesses equal protocol distributions. The IP protocol in version 6 still has a small percentage of traffic because most network hardware has not yet been designed for it. A common approach is tunneling of IPv6 in IPv4, but which is out of scope in this work.

Tables 6.1 and 6.2 show the protocol distributions which are mapped to deterministic and to random simulation traffic. It is assumed that the system environment supports the data link and physical layer types LLC/SNAP over Gigabit-Ethernet (G-E) and packet-over-SONET (POS). The order of packet transmission is indicated by letters that correspond to protocol-stacks. The explicit specification of a packet order only refers to deterministic traffic. In case of random traffic, the protocol distributions are identical, however the order is randomized.

[1] http://ipmon.sprintlabs.com, 17th march 2002

	Packets	Composition	
1	10000	90 % TCP-IPv4-G-E (A) 9 % UDP-IPv4-G-E (B) 1 % ICMP-IPv4-G-E (C)	Traffic distribution (1997) [131, 100] packet order: each 10th packet: B each 100th packet C
2	10000	82 % TCP-IPv4-G-E (A) 16 % UDP-IPv4-G-E (B) 2 % ICMP-IPv4-G-E (C)	Traffic distribution (2002) [101] packet order: each 16th packet: B each 50th packet C
3	10000	50 % TCP-IPv4-G-E (A) 49 % UDP-IPv4-G-E (B) 1 % ICMP-IPv4-G-E (C)	Equal application distribution packet order: A/B/A/B/... each 100th packet: C instead of B
4	10000	25 % TCP-IPv4-POS (A) 24 % UDP-IPv4-POS (B) 25 % TCP-IPv6-G-E (C) 24 % UDP-IPv6-G-E (D) 1 % ICMP-IPv4-POS (E) 1 % ICMP6-IPv6-G-E (F)	Equal distribution of IP versions packet order: A/B/C/D/... once per 100 packets: E instead of B once per 100 packets: F instead of D
5	10000	12 % TCP-IPv4-POS (A) 12 % UDP-IPv4-POS (B) 12 % TCP-IPv6-POS (C) 12 % UDP-IPv6-POS (D) 12 % TCP-IPv4-G-E (E) 12 % UDP-IPv4-G-E (F) 12 % TCP-IPv6-G-E (G) 12 % UDP-IPv6-G-E (H) 1 % ICMP-IPv4-G-E (I) 1 % ICMP-IPv4-POS (J) 1 % ICMP6-IPv6-G-E (K) 1 % ICMP6-IPv6-POS (L)	Equal packet distribution packet order: A/B/C/D/E/F/G/H/... once per 100 packets: I once per 100 packets: J once per 100 packets: K once per 100 packets: L
6	10000	12 % TCP-IPv4-POS (A) 12 % UDP-IPv4-POS (B) 12 % TCP-IPv6-POS (C) 12 % UDP-IPv6-POS (D) 12 % TCP-IPv4-G-E (E) 12 % UDP-IPv4-G-E (F) 12 % TCP-IPv6-G-E (G) 12 % UDP-IPv6-G-E (H) 1 % ICMP-IPv4-G-E (I) 1 % ICMP-IPv4-POS (J) 1 % ICMP6-IPv6-G-E (K) 1 % ICMP6-IPv6-POS (L)	Equal packet distribution with short bursts packet order: A/A/A/A/B/B/B/B/C/C/C/C/D/D/D/D/E/E/E/E/F/F/F/F/G/G/G/G/H/H/H/H/... once per 100 packets: I once per 100 packets: J once per 100 packets: K once per 100 packets: L

Table 6.1: Deterministic and randomized simulation traffic

6.2.2 Simulations and analysis

Misprediction rate

The misprediction rates for the first m_1 as well as for the first and second prediction stage m_{1+2} are defined as follows:

	Packets	Composition	
7	5000	90 % TCP-IPv4-G-E	Traffic distribution
		9 % UDP-IPv4-G-E	with burst
		1 % ICMP-IPv4-G-E	
	1000	100 % UDP-IPv4-G-E	
	4000	90 % TCP-IPv4-G-E	
		9 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
8	1000	90 % TCP-IPv4-G-E	Traffic distribution with 2 bursts
		9 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	1000	100 % UDP-IPv6-POS	first burst
	3000	90 % TCP-IPv4-G-E	
		9 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	1000	100 % UDP-IPv6-G-E	second burst
	4000	90 % TCP-IPv4-G-E	
		9 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
9	4000	90 % TCP-IPv4-G-E	Shifting traffic distributions
		9 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	2000	50 % TCP-IPv4-G-E	
		49 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	4000	19 % TCP-IPv4-G-E	
		80 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	2000	50 % TCP-IPv4-G-E	
		49 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	4000	90 % TCP-IPv4-G-E	
		9 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	

Table 6.2: Deterministic and randomized simulation traffic

$$m_1 = 1 - \frac{N_{h_1}}{N} \tag{6.1}$$

$$m_{1+2} = 1 - \frac{N_{h_1} + N_{h_2}}{N} \tag{6.2}$$

with the number of correctly predicted protocol-stacks in the first stage N_{h_1}, the number of correctly predicted protocol-stacks in the second stage N_{h_2} and the complete number of packets N. The number of correctly predicted protocol-stacks in the second stage implies that a misprediction occurred in the first stage. In Figure 6.2, the misprediction rates of the first prediction alone as well as of both, i.e. the first and the second prediction, are depicted.

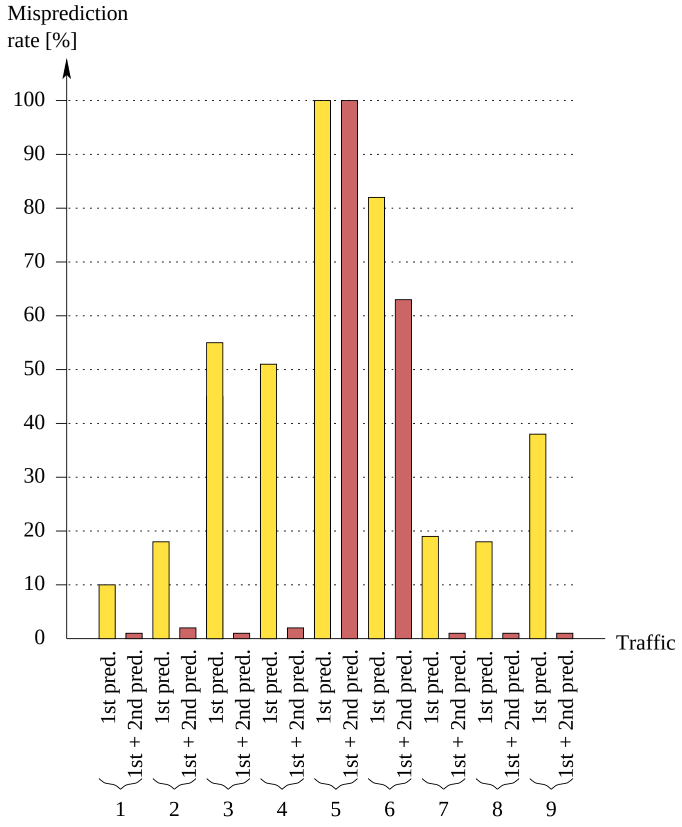


Figure 6.2: Misprediction rate

For this part of the analysis, the prediction weights are set to default value. For traffic 1 and 2, the misprediction rates for the first prediction stage are 10 % and 18 % respectively. However, the second stage achieves a hit rate of 90 % and 89 % respectively.

The hit rate for equal distributions of transport layer protocols, i.e. traffic 3, and for both, network and transport layer protocols, i.e. traffic 4, are significantly lower than for traffic 1 and 2. Even though, the miss rate is small at about 2 percent due to the second prediction stage.

The worst case is realized by traffic 5. The accumulated miss rates of 100 % for both stages result from the period order of packets. In case of equal frequencies of monitored packets which are used for prediction computation, the algorithm prioritizes the last received packet for MFU. Consequently, the predefined packet order leads to a period displacement of prediction table entries. In contrast, traffic 6 has the same protocol breakdown but a different packet order. As expected, two succeeding packets with identical SID exploit the history mechanism which was mentioned above. Thus, the failure rate of the first stage was reduced by 18 %.

Traffic 7 and 8 additionally comprise packet bursts. Compared to traffic 1, the first stage miss rate increases from 10 to 19 %. Nevertheless, large hit rates for the second stage of 95 and 94 % allow a failure rate for both stages of 1 %. The same miss rate can be achieved for traffic 9.

Modified weight factor

The weight factor has been defined in chapter 5.3 to priorize particular protocol-stacks.

The misprediction rates for modified prediction weights can be taken from Figure 6.3.

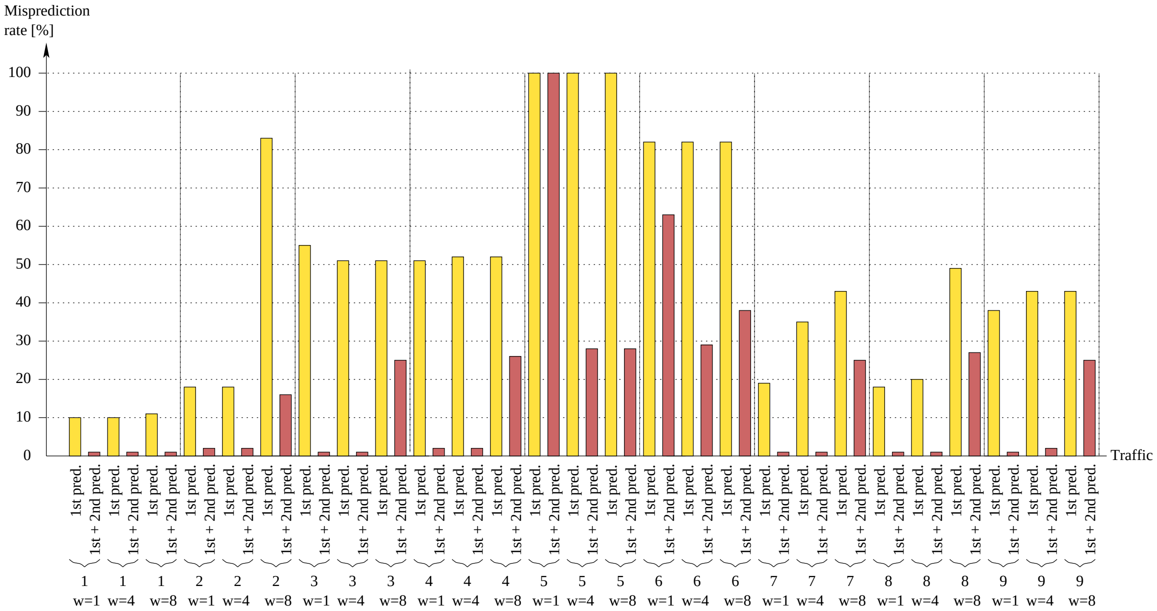


Figure 6.3: Misprediction rate, different weights

The weights in the figure are modified to priorize real-time traffic which relies on UDP. Apart from the default setting of all weights equal to $w[i] = 1$, the weights for UDP traffic have been exemplarily set to $w[i] = 4$ and $w[i] = 8$ for $i \in \{4..7\}$.

It can be seen that no alteration of prediction rates arises from w = 4 for traffic distributions with a noticeable majority of TCP traffic, i.e. traffic 1 and 2. For w = 8, the misprediction rate for traffic 2 is significantly increased.

The misprediction rates can be decreased for traffic 5 and 6 through modified weights. This can be explained by the priorization of UDP packets despite equal frequencies for the supported protocol-stacks.

The increase of the misprediction rate for traffic 7 and 8 results from the burst of UDP packets. While the accumulated misprediction rate for all packets is increased by higher values for the corresponding weights, the hit rate of UDP packets can be increased by a modification of $w[i]$ as well. Figure 6.4 shows the hit rates for $w[i] = 1$ for all packets and for $w[i] = 8$ for UDP packets only.

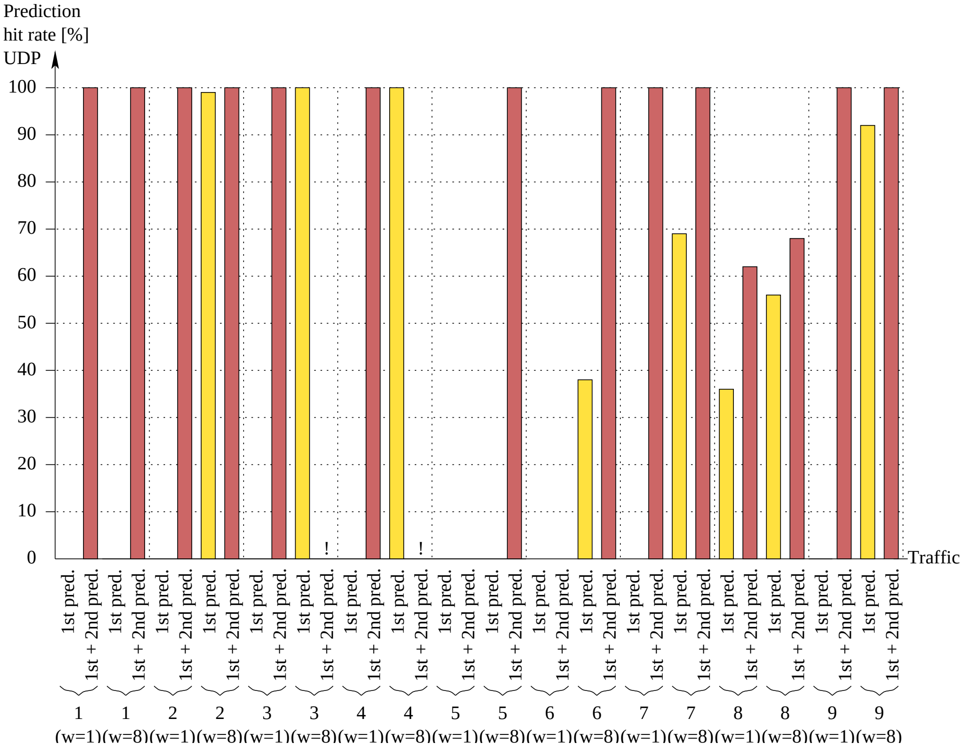


Figure 6.4: Hit rate UDP packets

It can be seen that the hit rate for a correct first prediction of UDP packets can be increased by $w[i] > 8$ for UDP protocol-stacks.

Initialization

Now, the probable impact of initialization packets on the prediction accuracy is analyzed. Initialization packets are used before start of operation to set up the prediction DHT. The origin of initialization packets might be taken from monitored traffic of an earlier system operation. After the DHT is set up, real user packets can be received. The observed results are presented in Table 6.3.

The obtained values demonstrate an unchanged behavior of prediction rates. It can be concluded that an initialization of the prediction DHT merely affects the prediction of the first received packets. A larger number of initialization packets has no impact on the prediction accuracy.

Traffic	Initialization Packets	hit rate 1st pred	hit rate 2nd pred	miss rate
2	10	0.82	0.160	0.02
2	20	0.82	0.160	0.02
2	50	0.82	0.160	0.02
4	8	0.492	0.488	0.02
4	16	0.492	0.488	0.02
4	40	0.492	0.488	0.02
9	10	0.6225	0.3675	0.01
9	20	0.6225	0.3675	0.01
9	50	0.6225	0.3675	0.01

Table 6.3: Impact of initialization packets

Processing Cycles

The mean number of processor cycles and the mean number of compare and branch instructions for the computation of updated prediction values is shown in Table 6.4.

Traffic	Mean number of computation cycles per packet	Mean number of compare/branch instructions per packet
1	331.1	27.9
2	340.7	29.4
3	381.1	35.9
4	376.5	35.8
5	417.8	41.8
6	395.7	38.0
7	342.4	29.7
8	340.8	29.7
9	365.8	33.5

Table 6.4: Computation cycles and number of compare instructions for prediction computation

The measurement has been done on a SUN SPARC Ultra-2 microprocessor that provides a reduced instruction set. One reason for this choice is the common approach to assign control-plane tasks in NPs to RISC processors. In this context, prediction computation is assigned to the control-plane as well.

It can be seen, that traffic 5 requires a mean number of 41.8 compare and branch instructions per packet. This can be explained by a higher number of processor register replacements for the cyclic traffic.

In order to evaluate the prediction algorithm in terms of computation effort, the number of compare and branch instructions is measured. These instructions are necessary to sort the protocol-stacks frequencies and to determine the predicted MFU entries. For this purpose, considered instructions are branch on equal (be), branch on greater (bg), branch on greater or equal (bge), branch on less (bl), branch on less or equal (ble), branch on not-equal (bne) and compare (cmp). For a predefined number of N = 16 supported protocol-stacks, the amount of compare and branch instructions is between $1.74 \cdot N$ and $2.61 \cdot N$. Compared to traditional sorting algorithms, which require between $\approx log(N) \cdot N$ and $\approx N^2$, a reduced computation effort is required by the applied prediction algorithm.

Dynamic behavior

The dynamic behavior of the prediction algorithm has been simulated with randomized traffics that correspond to the protocol distributions of Tables 6.1 and 6.2. However, their packet orders differ from the deterministic traffics. Figures 6.5 illustrates the dynamic behavior for traffic 1.

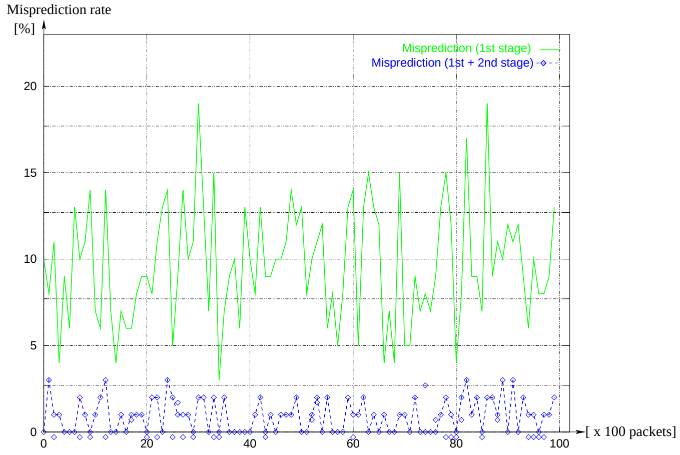


Figure 6.5: Traffic 1

The unit of the x-axis refers to 100 simulation packets. The number of first mispredictions ranges between 3 and 19 %. The upper bound for both predictions is about 3 %.

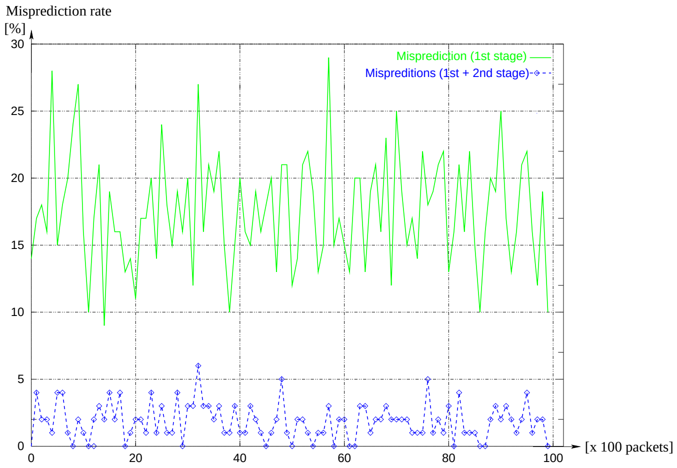


Figure 6.6: Traffic 2

In Figure 6.6, the captured sequence for traffic 2 is depicted.

With a change of protocol percentages towards equal distributions, the variation for the prediction increases. For instance, the variation of the first stage for traffic 1 and 2 is increased by 3 %.

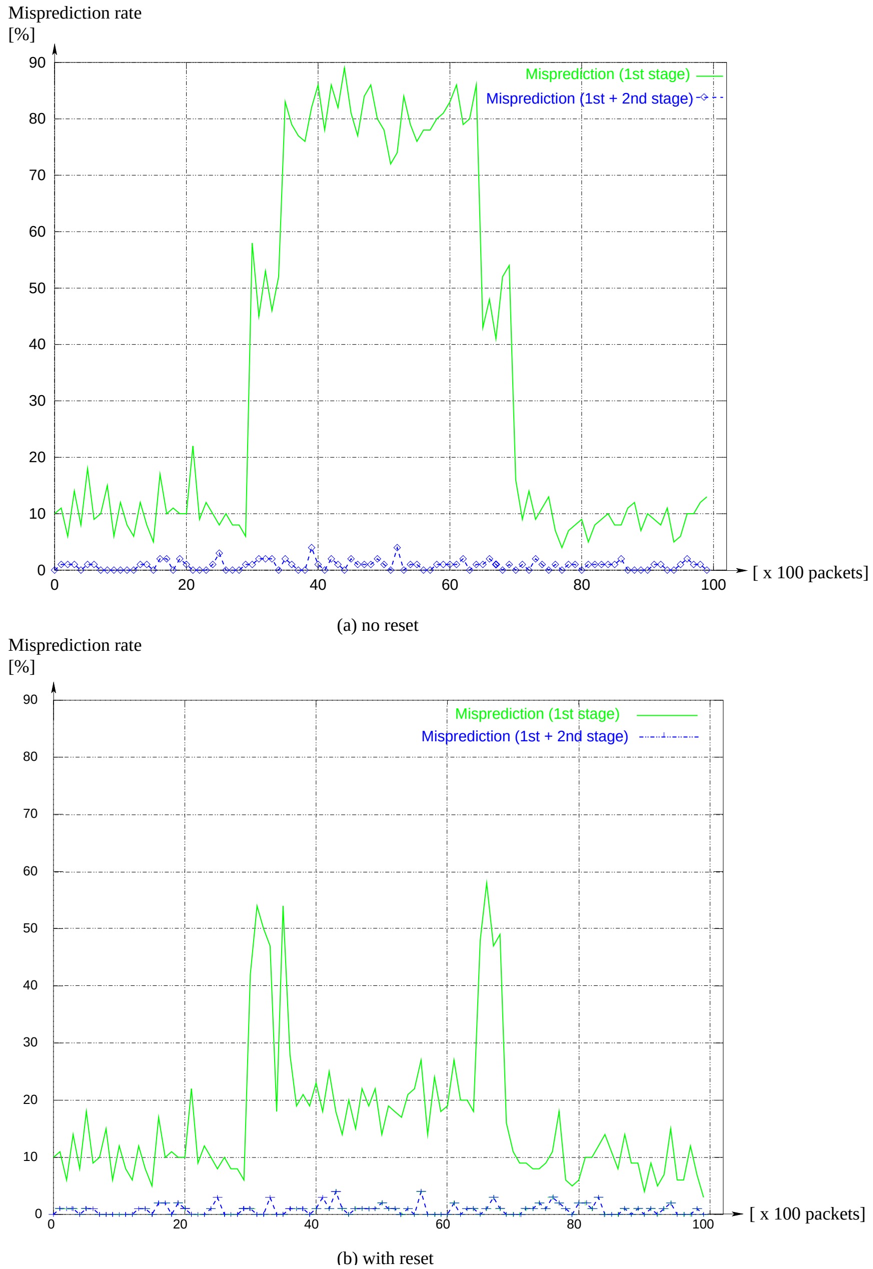
Prediction table reset facility

In order to achieve a fast adaptation to changed traffic statistics, the prediction algorithm is extended by a facility that periodically resets the prediction DHT table. For the evaluation, the randomized simulation traffic in Table 6.5 has been used. The simulation traffic 10 possesses two transitions of protocol distributions, the first after 3000 packets, the second after 6500 packets.

The misprediction rates of the prediction without and with the reset mechanism are shown in 6.7(a) and (b), respectively. Note that this extension is applied in this scenario to illustrate its benefit. The previous results shown above refer to the prediction without this table reset.

Figure 6.7(a) shows the dynamic behavior without table reset. It can be seen that a rapid increase for the misprediction rate from about 9 to 80 % occurs after 3000 packets. There is no decrease to smaller values until the distribution changes to the initial distribution. As expected, accumulated packet reception events, which are accumulated in the prediction table, prevent the prediction from a fast adaptation to changed distributions.

Figure 6.7(b) illustrates the results for the algorithm which exploits the table reset capability. In order to achieve a fast prediction adaptation to changed distributions, the reset period is predefined to n = 500 received packets. After the first transition, i.e. after 3500 packets, the misprediction rate for the first stage decreases significantly to about 19 %.



(a) no reset

(b) with reset

Figure 6.7: Prediction table reset facility

	Packets	Composition	
10	3000	90 % TCP-IPv4-G-E	Shifting traffic distributions
		9 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	500	50 % TCP-IPv4-G-E	
		49 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	3000	19 % TCP-IPv4-G-E	
		80 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	500	50 % TCP-IPv4-G-E	
		49 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	
	3000	90 % TCP-IPv4-G-E	
		9 % UDP-IPv4-G-E	
		1 % ICMP-IPv4-G-E	

Table 6.5: Randomized simulation traffic

The exemplary choice for x = 30 and y = 10 (chapter 5.3) can be illustrated by the following assumption: The first protocol distribution for 3000 packets is 90 % TCP-IPv4-G-E, 9 % UDP-IPv4-G-E and 1 % ICMP-IPv4-G-E, followed by a second of 3000 packets with 19 % TCP-IPv4-G-E, 80 % UDP-IPv4-G-E and 1 % ICMP-IPv4-G-E. If the number of received packets corresponds to the mean value during the first 3000 packets, 2700 packets of TCP-IPv4-G-E, 270 packets of UDP-IPv4-G-E and 30 packets of ICMP-IPv4-G-E have been received. If the complete prediction table would be reset and x as well as y are set to 0, a mean number of 2454 packets are necessary to change the prediction from TCP-IPv4-G-E to UDP-IPv4-G-E during the second distribution. If the table is reset, but x is set to 30 and y is set to 10 instead, the mean number of received packets that are necessary to change the prediction is merely 33.

6.3 Packet-processing element

6.3.1 System implementation

The main intention of an implementation is the functional verification and performance simulation of a PPE. For the description language that enables system-level modeling, SystemC is used. In the first part of this chapter, a brief introduction of system-level design is given. In the second part, the simulation environment is explained followed by the obtained results.

System-level design flow

According to [133], the system-level design flow contains the steps depicted in Figure 6.8.

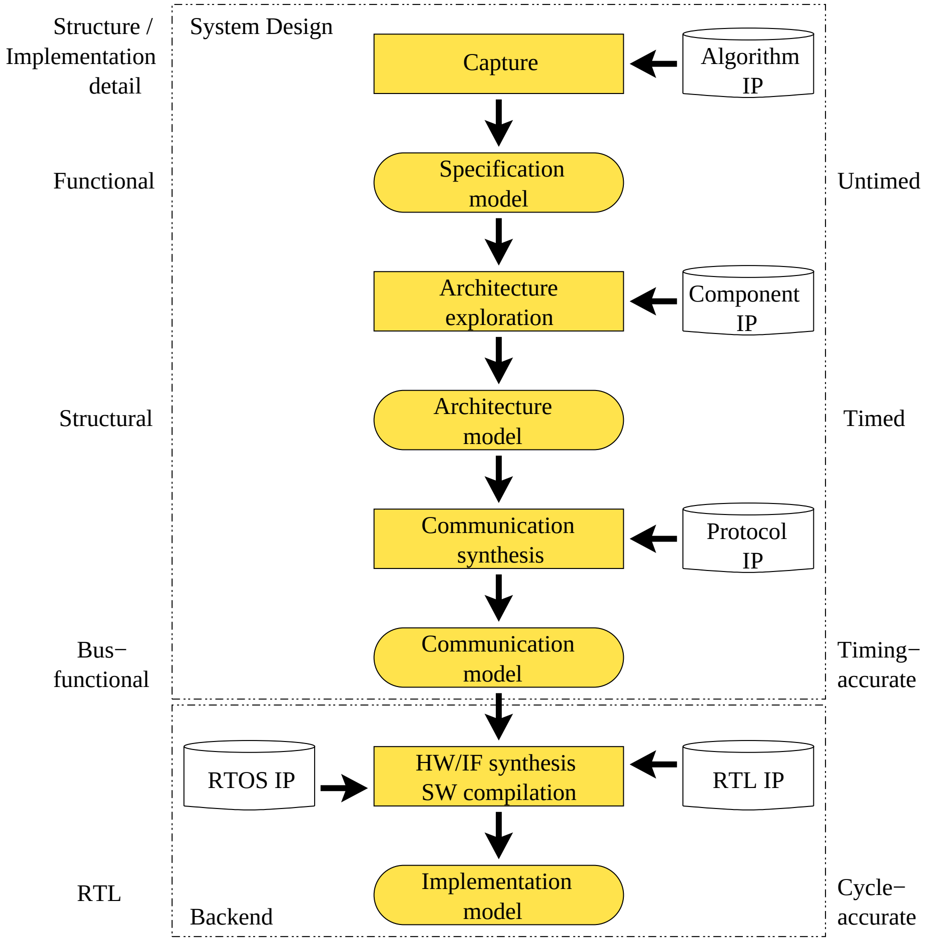


Figure 6.8: System-level design flow

The approach comprises multiple layers of abstraction: the pure functional, the structural and the bus-functional layer. The design starts with the development of a specification model that is captured by the user. At the architecture level, the structure of the system architecture is defined under consideration of a corresponding component library. The functional description is divided into several partitions. Each of them is mapped to an architecture component. During communication synthesis, components are refined into bus-functional representations which provide a timing-accurate communication model. Finally, the backend covers hardware, software and interface synthesis through an enhancement of timing granularity of the components. The intellectual property (*IP*)-modules are used to achieve a reduction of design effort and consequently to decrease the design time.

Currently, two of the most relevant approaches for system-level design languages are SpecC [134] [135] and SystemC [136] [137]. Both are based on C/C++. They allow a description of parallelism, structural and behavioral hierarchies as well as synchronization and communication. Both furthermore provide the deployment of simulation models that refer to one of the abstraction levels in the figure above.

SystemC

According to [138], there are two modeling concepts for SystemC: the *classical hardware modeling* and the *functional modeling*. With "classical hardware modeling" modeling started at register-transfer level and behavioral level is meant. "Functional modeling" instead allows a description of system components while the partitioning into hardware and software components is still out of scope. At that time, aspects such as timing, fine-grain architectural structures or low-level communication protocols were not considered. In order to validate system concepts before the implementation, functional models of the system can be used. The two kinds for functional models, are *untimed* and the *timed*. They merely differ in the annotation of timing information to the model.

Thus, the capability for functional modeling, the availability of the SystemC core language and the communication and synchronization modeling features in version 2.0 were essential reasons for the application of SystemC for system evaluation. In addition, the potential strength of the academical and industrial participants of the open SystemC initiative (OSCI) appears to support SystemC to become the state-of-the-art standard for system-level description languages. Participants of OSCI are EDA vendors, e.g. Synopsys, Cadence, Mentor Graphics and CoWare [139].

Implementation

Packet-processing element

Apart from the shared bus, the PPE architecture is divided into master and slave modules. Master processing modules, which are shown on top, can be characterized by the capability of process initiation and response on requests. This comprises the network layer PUs, the control-point and the I/O buffers. In contrast, slave modules merely respond to master requests. Figure 6.9 illustrates this classification in a PPE.

Two internal communication modes exist. Control signaling between master modules is done with a master-master-message (MMM) bus . For data transfer between two masters or a master and a slave module, the address and the 32-bit data bus are used. The access to the bus is granted by a bus arbiter.

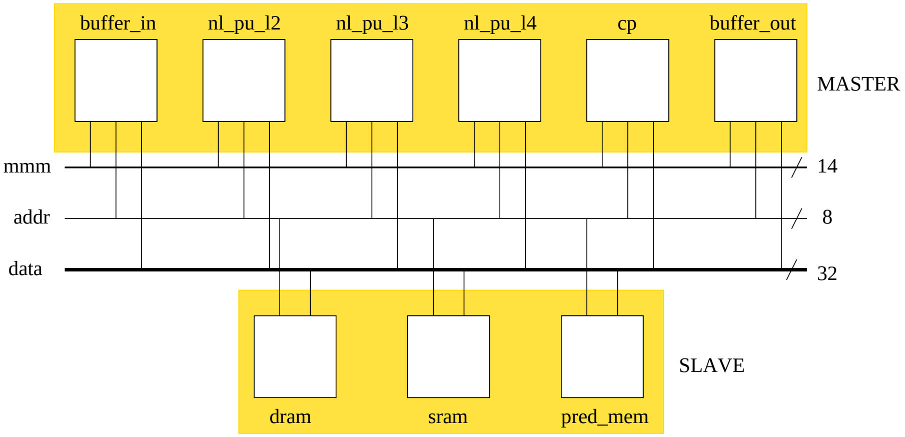


Figure 6.9: PPE master and slave modules

Protocol-processing

Protocol-processing takes place in separate NL-PUs which are realized as microprocessor cores. The supported protocol-processing tasks are implemented as instruction sequences that rely on a predefined RISC-instruction set. The functional implementation of the protocol-processing flow in a NL-PU is shown in Figure 6.10.

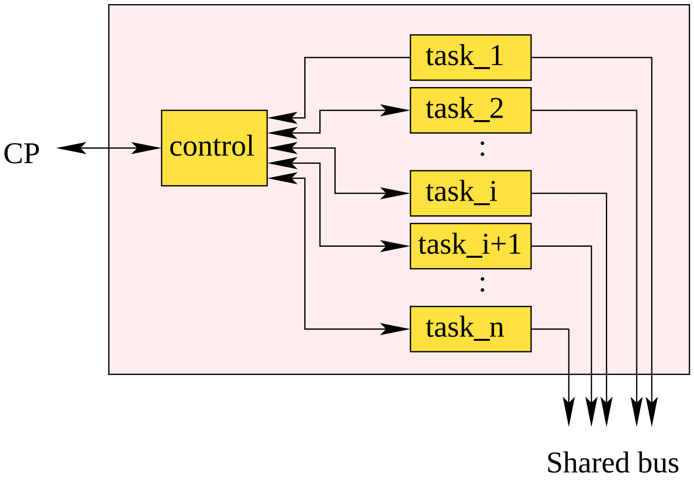


Figure 6.10: Protocol-processing flow

Protocol-processing tasks are initiated by a control unit that is responsible for task sequencing and execution control. After completion of a task i is signalized to the control unit, the succeeding task i+1 is invoked. Each task can comprise one or several accesses on shared resources, i.e. memory devices. The starting point for the embedded processor is a single-scalar architecture. Thus, only one access on shared resources can be done at a time. The completion of the last task and processing malfunctions are indicated to the CP. The implementations of embedded processors, memory and busses furthermore provide additional monitoring signals which are used to evaluate their workload. The protocol-processing flow depends on the results of conditional instructions. In this system design, conditional branches are limited to the content of protocol-layer type fields, availability of optional header fields and destination address fields. The flow of remaining tasks for a packet follows the predefined protocol-stack. The mean computation time for the single tasks is derived from their RISC implementation on the Intel IXP 1200 NP [75]. Arbitration, bus and memory access times are predefined as well. Their mean durations can be derived from [140] [141] [75]. In case of a shared bus architecture with a maximum of four simultaneous bus requests, compared to the IXP1200, the bus structure can be realized with a lower complexity. Thus, lower delay values compared to the IXP 1200 are assumed as shown in Table 6.6.

Device	Instruction	Processor cycles
SDRAM	read, write	25
SRAM	read, write	10
Prediction DHT	read	1
Arbiter		2

Table 6.6: Memory access times

Memory clock frequencies are commonly lower than processor frequencies [75]. Note that the cycles in the table refer to the processor clock frequency, which is assumed to be twice the memory clock frequency.

6.3.2 Test methodology and constraints

Simulated systems

In order to determine the protocol-processing performance, the simulated system is restricted to a PPE. This can be done based on the predefinition of separate memory devices for the PPEs in the NP. The simulated systems can be seen in Figure 6.11.

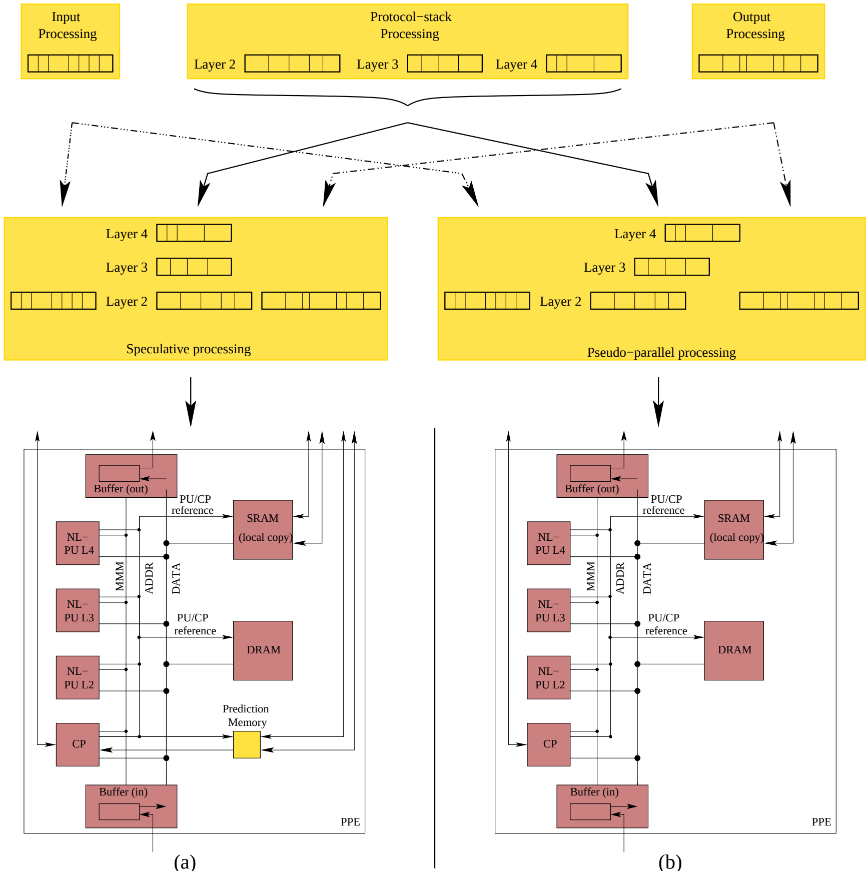


Figure 6.11: Simulated systems

Protocol-stack tasks are mapped to the underlying architecture according to the corresponding processing model, i.e. speculative (Figure 6.11(a)) and pseudo-parallel (Figure 6.11(b)). The pseudo-parallel model is used as a reference for the evaluation. The input and output processing flows are identical in both systems. Input processing represents the processing flow after the reception of a packet by the PPE and before the initiation of protocol processing, e.g. PDU data transfer to DRAM. After protocol-processing, output processing comprises the assembly of layer-specific PCI with the stored PDU and the data transfer to subsequent output queues. The assignment of protocol-layer tasks corresponds to the layer numbers of the NL-PUs. QoS tasks, i.e. classification, policing and accounting, are performed by NL-PU L4.

The PPEs in both systems follow the predefinition that only one packet per time is processed. The processing model of a PPE is shown in Figure 6.12

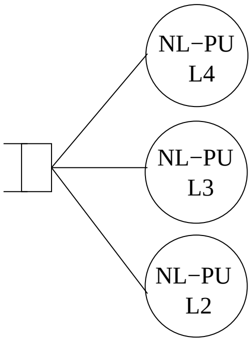


Figure 6.12: PPE processing model

Apart from synchronization tasks which are performed by the CP, the processing model corresponds to a queuing model that consists of a single queue and three servers. Each of them executes assigned protocol tasks of the corresponding network layer. Based on this, single NL-PUs can be idle before each of them has completed protocol-processing and the next packet is dispatched.

Simulation traffic

The randomized simulation traffic is shown shown in Table 6.7.

6.3.3 Simulations and analysis

Based on the model predefinitions above, the simulation results will be presented and discussed. In case of a static prediction, the observed results can partially be analytically derived, too.

Latency

For the determination of the mean protocol-processing latency, TCP-over-IPv4-over-G-E is exemplarily applied as predicted protocol-stack for the first prediction stage. The latencies per packet are shown in Table 6.8.

Traffic	Composition	
1	90 % TCP-IPv4-G-E 9 % UDP-IPv4-G-E 1 % ICMP-IPv4-G-E	Traffic distribution (1997) [131, 100]
2	82 % TCP-IPv4-G-E 16 % UDP-IPv4-G-E 2 % ICMP-IPv4-G-E	Traffic distribution (2002) [101]
3	50 % TCP-IPv4-G-E 50 % UDP-IPv4-G-E	
4	50 % TCP-IPv4-G-E 50 % UDP-IPv6-G-E	
5	50 % TCP-IPv4-G-E 50 % UDP-IPv4-PoS	
6	80 % TCP-IPv4-G-E 20 % UDP-IPv4-G-E	
7	100 % TCP-IPv4-G-E	

Table 6.7: Simulation traffic

Received packet protocol-stack	Number of mispredictions	speculative model [processor cycles]	pseudo-parallel model [processor cycles]
TCP-IPv4-G-E	0	841	1164
UDP-IPv4-G-E	1	1168	1150
UDP-IPv6-G-E	2	1452	1293
UDP-IPv6-POS	2	1478	1315

Table 6.8: Mean protocol-processing latency

The mean values refer to a series of identical protocol-stacks that are received in a sequential order by the PPE. The table reveals that only in case of no mispredictions, the speculative processing model exhibits a smaller value for the latency than the pseudo-parallel. The difference between both processing models is 323 computation cycles, which corresponds to a delay reduction of 27.7 %.

Based on an identical and static prediction for the speculative model, the mean latencies in Figure 6.13 result from simulations with the randomized traffic stimuli of Table 6.7.

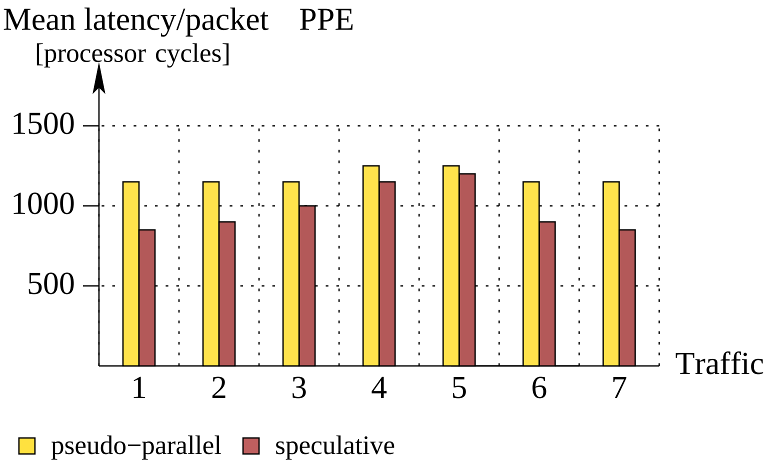


Figure 6.13: PPE latency

The latency reduction can be derived with equation 4.1 as shown in Figure 6.14.

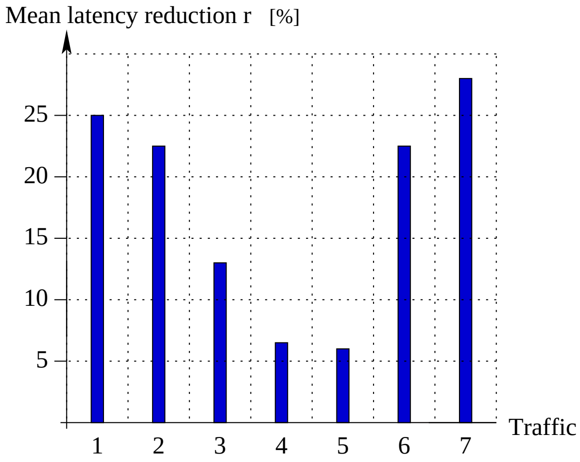


Figure 6.14: Latency reduction

A maximum reduction of r = 27.7 % can be achieved for traffic 7, i.e. in case of no mispredictions. For current traffic characteristics, i.e. traffic 2, the value is about 22.5 %. The reduction decreases with equal packet distributions to r = 6.4 %.

The effort for speculative protocol processing in terms of processor cycles is higher than 841 cycles for the chosen constraints. For a computation of prediction values based on a RISC processor, between 331 and 417 cycles were necessary in chapter 6.2. Consequently, dynamic tables updates at line-speed are feasible. In case of multiple PPEs per system, an additional aggregation of received SID has to be done before the update of prediction values.

Individual processor workload

The costs in terms of embedded processor workload are depicted in Figure 6.15. The workload represents execution times when NL-PUs execute protocol tasks. Based on the predefinition of three servers, i.e. NL-PUs, a higher processor workload does not require additional resources, e.g. through an assignment of external processors.

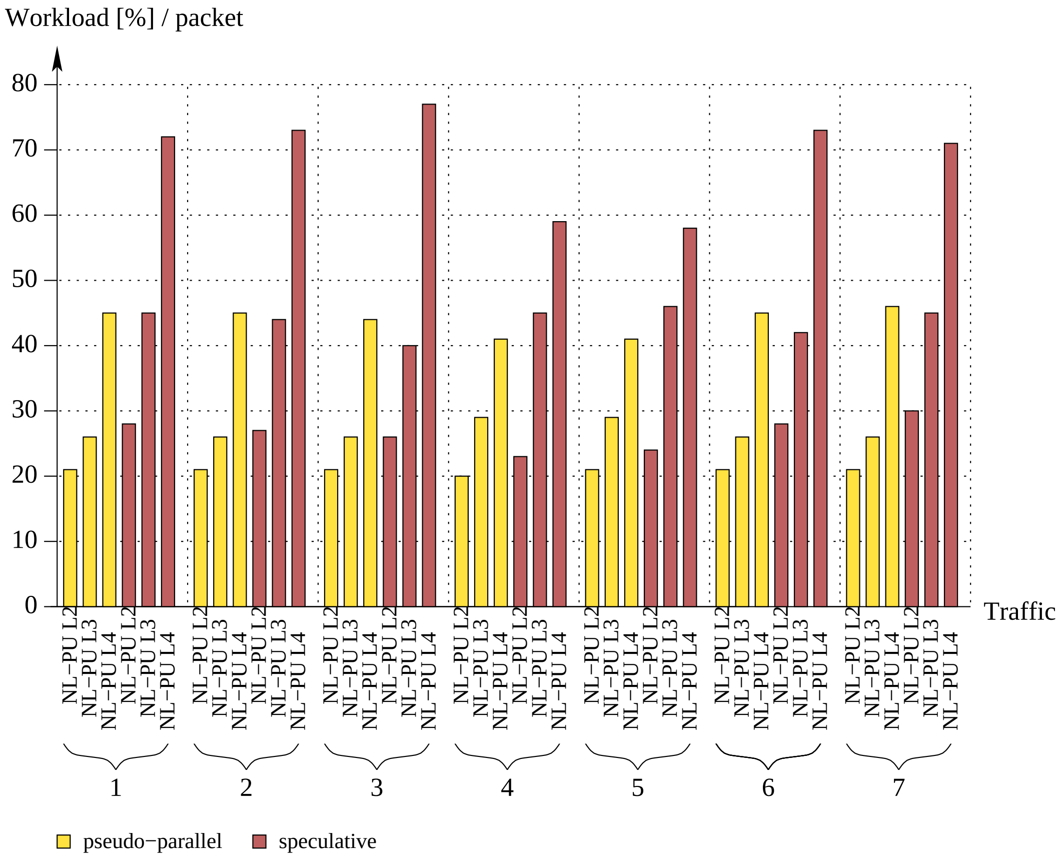


Figure 6.15: Individual processor workload

The relative quantities refer to the complete duration that is required for protocol-processing. Thus, a comparison of workloads for different traffic has to account for the corresponding protocol processing delays. The figure reveals that the workload for speculative processing is higher than for pseudo-parallel execution, resulting on one hand from an additional effort in case of mispredictions. On the other, during memory accesses program execution in scalar processor stalls until reference termination. Consequently, the processor is busy. High values are especially obtained for the NL-PU L4 which performs the time-consuming DiffServ tasks, i.e. classification, policing and accounting.

The correctly predicted traffic 7 demonstrates the unbalanced workload of the NL-PUs.

While NL-PU L2 is busy for only 252 processor cycles, NL-PU L4 requires 597 cycles. A latency reduction for speculative protocol-processing can be expected if the workload is balanced, i.e. if tasks of NL-PU L4 are assigned to NL-PU L2 and NL-PU L3.

Aggregated processor workload

The aggregated values for the three NL-PUs can be taken from Figure 6.16.

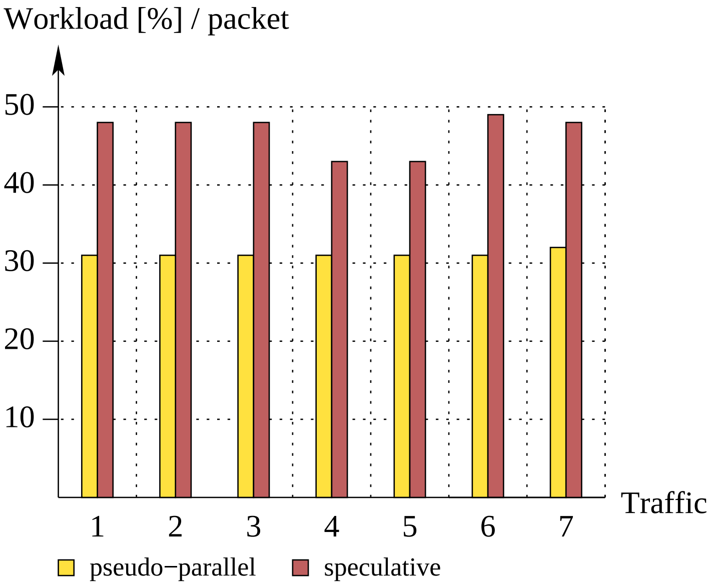


Figure 6.16: Aggregated processor workload

While the deviation for pseudo-parallel processing is small, the added workload for speculative processing is between 43 % and 49 %. For traffic 2, the additional workload is about 55 %. The workload decrease for traffic 4 and 5 results from the fact that relative quantities are depicted.

Arbitration- / Bus- and Memory-Load

The access to local memories via the shared bus is granted by a bus arbiter. The workloads of the arbiter, the shared bus as well as the memory devices are presented in Figure 6.17.

It can be observed that the accesses to the DRAM for the speculative model achieve a maximum value of 11 %. These higher values result from additional memory load instructions which are necessary for partial PCI reload in case of speculatively overwritten content in NL-PU registers.

The number of SRAM accesses is increased in case of speculative processing. This results especially from routing, classification and policing tasks that are wrongly executed as a consequence of mispredictions.

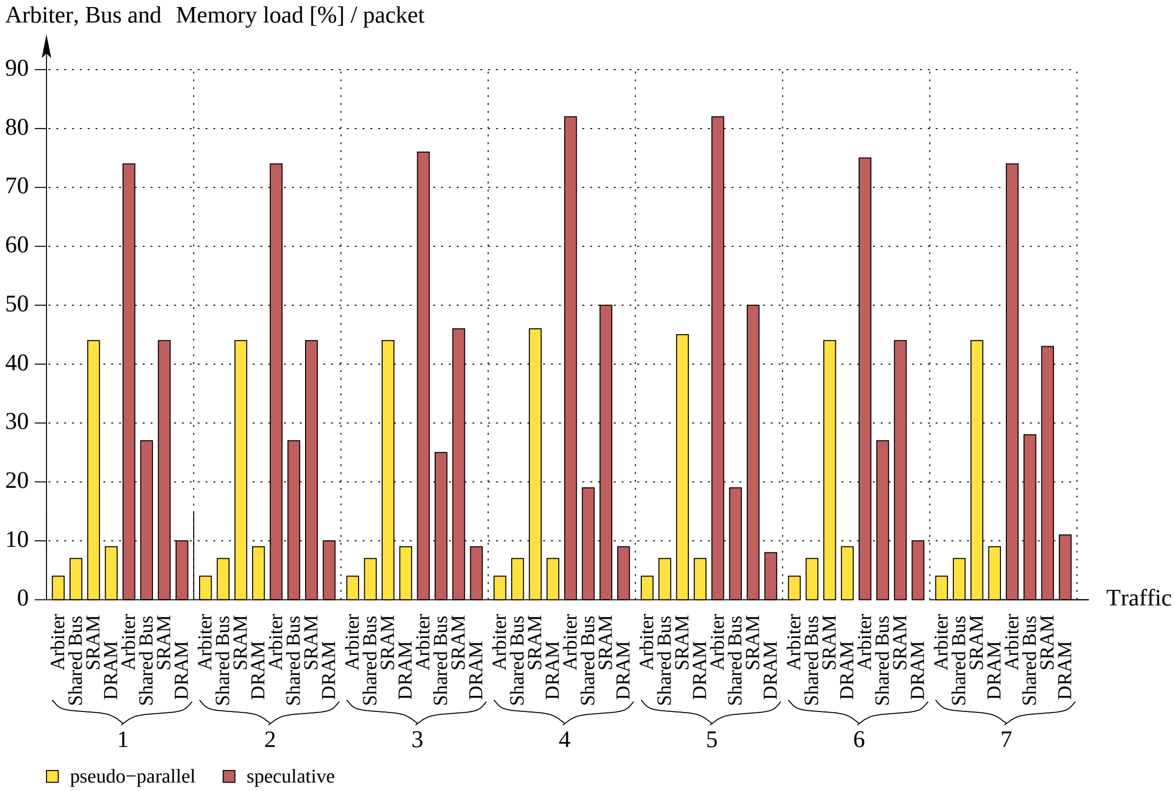


Figure 6.17: Arbitration- / bus- and memory-load

An essential outcome of the simulation is a significant difference for arbitration requests for shared memory resources. While memory accesses in case of pseudo-parallel processing are spread over the complete processing time of a packet, initiation of layer processing in parallel leads to a higher number of synchronous bus requests. This means that the shared bus arbiter represents a limitation for the system performance. As a consequence, stall times increase the processor workload which has been already shown in Figure 6.16. Simultaneous requests to the MMM-bus occur rarely in relation to the complete processing time of a packet. Consequently, this interconnection bus between embedded processors of a PPE does not appear as a performance bottleneck.

Throughput

In order to determine the throughput of a single PPE, two basic assumptions are made: Considering current NP properties, the processor clock is set to 400 MHz [142]. A unified minimum packet size of 96 bytes is used as simulation stimuli for all supported packets. The throughput results are depicted in Figure 6.18.

It can be seen that the throughput of the speculative model is higher than that of the refer-

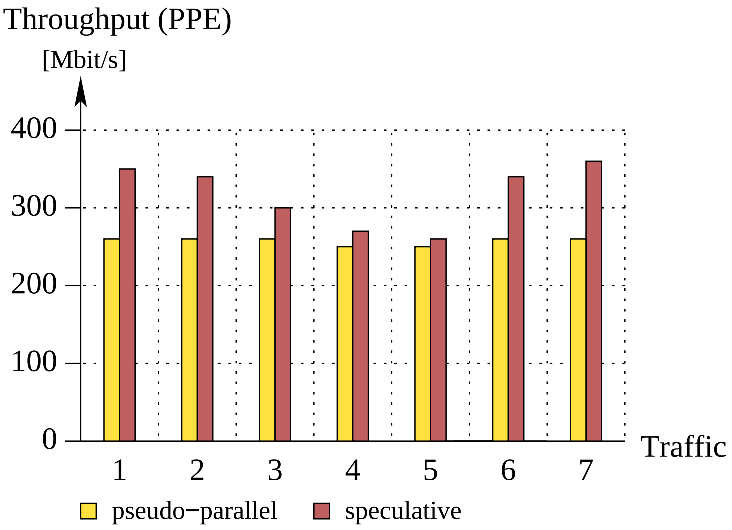


Figure 6.18: PPE throughput

ence implementation. This results from a higher latency for protocol-processing of a single packet. It has to be mentioned however that the obtained values refer to the predefinition of one processed packet per time in a PPE. The idle times in NL-PUs consequently exhibit that the system is over-provisioned in terms of processor resources.

The system performance depends on the entire number of processing resources to which protocol-tasks can be assigned to. In case of a non-over-provisioned system, a maximum amount of processing resources is predefined. If the processing requirements for protocol-processing of a packet exceed this value, the execution of several tasks stalls until processing resources requirements fall below this limit l. Consequently, the processing delay is increased.

The value for l has to be specified before operation. The value that is assigned to l corresponds to the maximum workload per packets. In case of the two applied models, i.e. the speculative and the pseudo-parallel model, the highest workload per packet for both models is given in the speculative case if every received packet obtains two mispredictions, i.e. one in each prediction stage. This value is specified as l. The exact value depends on one hand on the protocol-stacks which are supported by the system. On the other hand, it has to be noted that a strict relation for the workload exists between received protocol-stacks and the currently predicted stacks of the dynamic prediction algorithm.

Considering the simulation traffic in Table 6.7, UDP-IPv6-G-E is used as the protocol-stack that specifies the maximum value for the required workload. It is supposed that the prediction leads to 2 mispredictions for the corresponding packet. The workload in cycles for this protocol-stack is shown in Table 6.9. The index i in the table refers to the protocol layer.

	Workload (speculative model) [cycles]	rel. Workload (speculative model) $w_{s_{L_i}}$ [%]	Workload (pseudo-p. model) [cycles]	rel. Workload (pseudo-p. model) $w_{p_{L_i}}$ [%]
Latency	1452		1315	
NL-PU L4	757	0.52	477	0.36
NL-PU L3	654	0.45	411	0.31
NL-PU L2	276	0.19	248	0.19

Table 6.9: NL-PU workload for UDP-IPv6-G-E

Both models differ in the values for the processing latency. If the processing delays are normalized for both models, the additional NL-PU resources for the pseudo-parallel model can be determined. Considering the values in the table above, the workload in case of pseudo-parallel processing is decreased by

$$\begin{aligned}(w_{s_{L_4}} - w_{p_{L_4}}) + (w_{s_{L_3}} - w_{p_{L_3}}) + (w_{s_{L_2}} - w_{p_{L_2}}) &= \\ (52\% - 36\%) + (45\% - 31\%) + (19\% - 19\%) &= 30\%\end{aligned} \tag{6.3}$$

6.4 Conclusion

The evaluation presented in this chapter covered the prediction unit as well as the PPE. Separate simulation models, a model in C/C++ for the prediction unit and a model in SystemC for the PPE, were applied to focus on one hand on the behavior of the prediction algorithm and on the other hand on communication aspects of the shared bus architecture. Both parts have been simulated with real packet distributions of 1997 and 2002. In addition, artificial protocol distributions have been used for worst-case analysis. Since the simulation of the PPE covers worst-case traffic compositions, the implementation of a static prediction module in the model does not constrain the quantity of the obtained performance results.

In the first part, the dynamic prediction algorithm has demonstrated a high accuracy and an efficient adaptation to varied traffic distributions. The necessity for a small amount of prediction registers and a small mean number of compare instructions highlighted the strength of the chosen algorithm.

The implementation of a PPE architecture was focused on a performance evaluation of the underlying processing model. Consequently, the obtained results refer to the predefined system environment and to the simulation constraints. These values have shown that a significant reduction for the processing latency can be obtained through speculative processing. For this processing model however, data access to memories was impeded by a higher number of simultaneous requests for shared bus resources. The assignment of protocol-layer tasks to the NL-PUs has an impact on the processing latency. If the processor workload is unbalanced, the NL-PU with the highest workload determines the processing latency.

The exemplary NP architecture represents an over-provisioned system. In case of a non-over-provisioned system, the amount of assigned processor resources to a packet is limited. During the system simulations, the maximum workload was required in case of speculative processing for 2 occurring mispredictions. If this value is specified as the maximum processor workload for protocol-precessing of a single packet, available processor resources become idle in case of a lower number of misprediction. If the pseudo-parallel reference model also provides an identical amount of processor resources, spare processing capacities are given.

Chapter 7

Concept application study

7.1 Introduction

The objective of the speculative packet-processing concept is to achieve a reduced latency for protocol-processing compared to traditional processing models. Low latencies are relevant for delay-sensitive real-time (RT) applications that possess bounded maximum end-to-end transmission delays for each packet [143]. If a packet exceeds this deadline, i.e. it arrives too late at the destination, its value is reduced or it is even useless for the application. Besides low latency, an important property for RT communication is a low delay jitter. Several studies have been done to provide bounded delay jitter in packet-switching networks [144] [145].

However, results of section 6.2 show that the delay in case of speculative processing depends on the protocol-stack prediction. While a correct prediction achieves low latencies in a PPE, one or two mispredictions entail longer processing delays. For a data stream that is transmitted in several packets, a delay jitter can arise from these different values of the network node delay. In order to compensate increased mean values for processing delays and delay jitters of packets, an appropriate concept will be introduced in section 7.2. In the following section, the benefit of the speculative processing concept will be analyzed from an end-to-end perspective. Finally, the complete latency reduction along the transmission path will be estimated under consideration of network node and network domain properties.

7.2 Queuing delay and jitter analysis

7.2.1 Motivation

In Table 6.8, the additional processing time in case of occurring mispredictions was shown. While one misprediction entails an additional processing time consumption of more than

300 cycles, two mispredictions may require about 600 processor cycles more than in case of none mispredictions. In order to cope with these deviations, a delay compensation has to be achieved subsequently to the PPEs. Separate packet processing paths for the input and output are used in NPs that have a switch fabrics between, e.g. [146]. In Figure 7.1, a centralized architecture is shown that performs protocol-processing and output processing in a single system, e.g. [21].

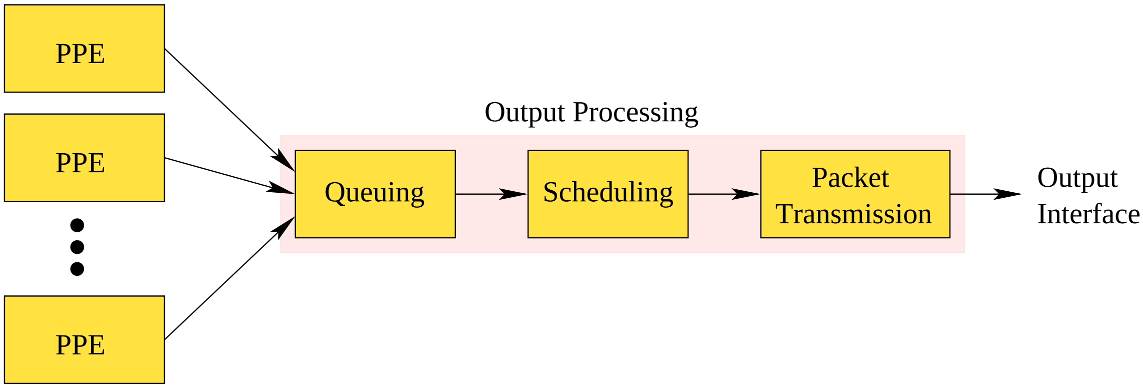


Figure 7.1: Output data movement model

Packets which have finished protocol-processing are transferred to the output processing block. This block is responsible for enqueuing and dequeuing of packets. A scheduler decides the service order for packet transmission to the outgoing interface. Further functionalities, e.g. congestion avoidance mechanisms, might be implemented here as well, but they are out of the scope of this work. Finally, packet transmission prepares packets according to the internal data structure for transmission.

7.2.2 Scheduling algorithm

The targeted delay jitter compensation is realized by a scheduling algorithm that has the capability to assign higher queuing priorities to mispredicted packets. One commonly implemented scheduling algorithms in todays routers is WFQ. Figure 7.2 shows an extension of the WFQ algorithm which additionally has two priority queues (PQ) [147] [148]. It is called PQ+WFQ hereafter.

Packets which refer to different classes and receive different forwarding behavior are stored in different queues based on the type-of-service field and traffic-class field of the IP version 4 and IP version 6 header, respectively.

For an individual prioritization of single WFQ queues, queuing weights w_i are defined. They allow an allocation of individual bandwidths and delays for traffic flows. For simplicity reasons, two per-hop behaviors are used in the following: Expedited Forwarding (EF)

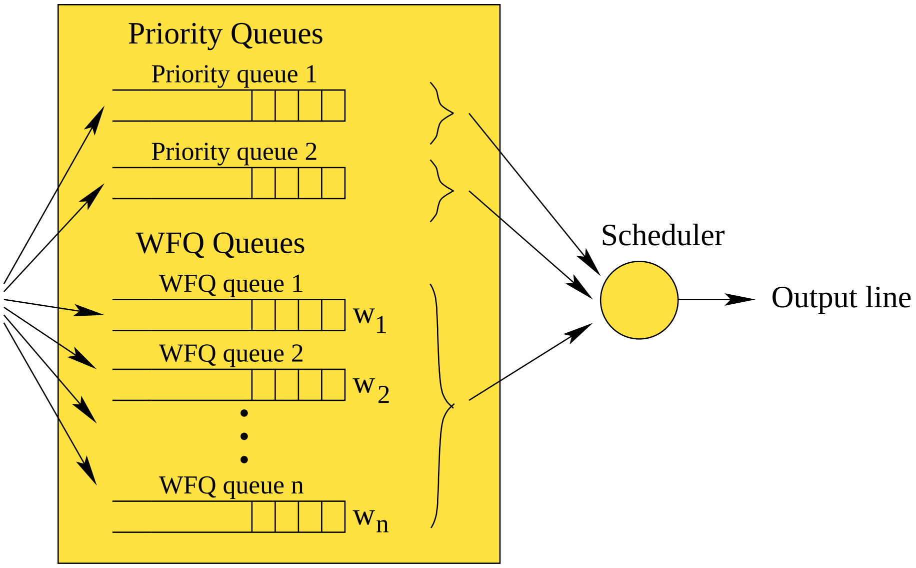


Figure 7.2: Priority + weighted fair queuing

and Best-Effort (BE). Large values for w_i are assigned to EF queues, which contain delay- and jitter-sensitive packets. BE traffic is kept in queues possessing small values for w_i.

The priority queues are used for EF packets that are delayed due to mispredictions. In case of one misprediction, the packet is stored in priority queue 2 and in case of two mispredictions in priority queue 1, respectively. The number of occurred mispredictions is indicated by the prediction history word (PHW) which is delivered simultaneously to the packet data by the precedent PPE. Packets with no mispredictions during protocol processing are distributed among WFQ queues that correspond to the CoS label of the IP header. The number of WFQ queues is exemplarily defined to four. The scheduling algorithm traverses down the queues and serves successively priority queue 1 and 2 if a packet is stored. Otherwise, the scheduler serves the WFQ queues according to their weights. It is assumed that the scheduler has a work-conserving behavior, i.e. the scheduler is never idle when packets await service.

7.2.3 Evaluation

The evaluation is focused on the metrics mean delay and delay jitter. The observed variable *delay jitter* is defined as a quantity that expresses the variability of the delay. Considering several different definitions of delay jitter [149], it is applied in this context as the difference between the maximum and the minimum delay of packets in a network node [144].

In total, three simulation scenarios are observed. The reference system implements pseudo-parallel processing and weighted fair queuing. The speculative model represents an improvement of the processing methodology. However, mispredictions generate a delay jitter which can not be reduced by WFQ. Finally, PQ+WFQ is evaluated in conjunction with the speculative processing model. The simulation methodology is illustrated in Figure 7.3.

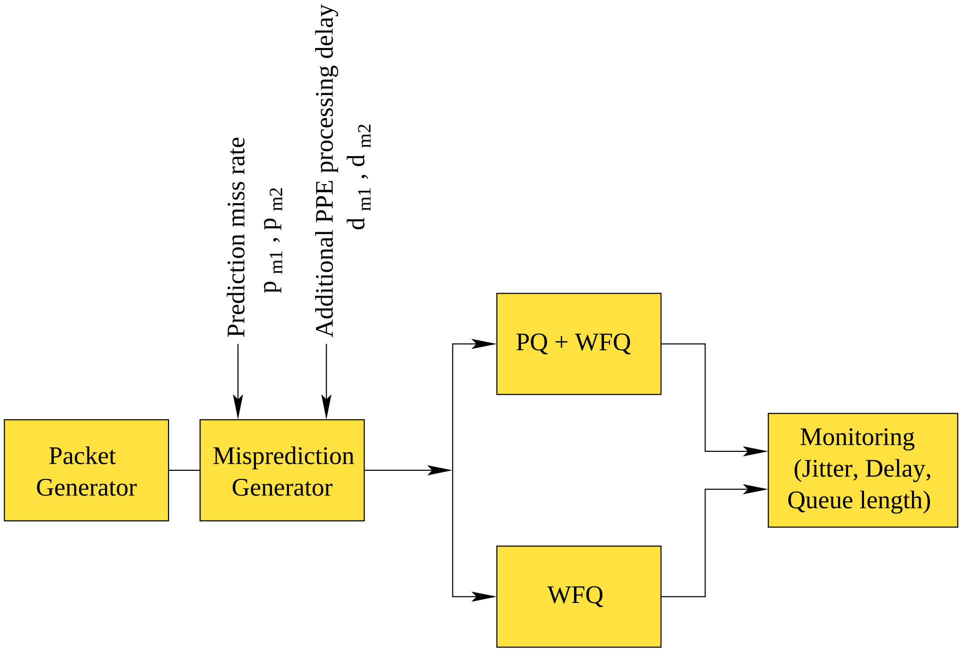


Figure 7.3: Evaluation Methodology

A packet generator is used to generate random simulation traffic for both queuing models. In order to consider processing delays resulting from the occurrence of mispredictions in PPEs, processing delays d_{m1}, d_{m2} are added to single packets according to the statistical rates p_{m1}, p_{m2} for one or two mispredictions. The indices refer to the number of mispredictions for a packet. In case of no misprediction, d_{m0} is equal to zero. According to Table 6.8, the additional processing delay in case of one misprediction is about $d_{m1} = 300$ cycles and in case of two mispredictions about $d_{m2} = 600$ cycles, respectively. The misprediction rates, derived from the traffic statistics in [101], are $p_{m1} = 2$ % for 2 mispredictions and $p_{m2} = 16$ % for one misprediction. As a fundamental requirement for the simulation, the queuing system has to be in steady state, i.e. the incoming and outgoing packet rates have to be equal. It has to be mentioned that only mispredicted EF packets are stored in priority queues, mispredicted BE packets are kept in WFQs. The size for simulation packets is 96 Bytes, which corresponds to the predefinition in the previous chapter. The outgoing link rate is set to 8 bit per cycle. In case of an NP clock frequency of 400 MHz, the transmission rate ranges between OC-48 and OC-192. The number of simulation packets is 1000. In order to avoid empty packet queues during simulation, two simulation periods are used. The incoming data rate is higher than the outgoing link rate during the initialization. The queues are filled with a number of predefined initialization packets n_i. During the operation time, the mean incoming data rate is equal to the mean outgoing link rate to avoid queue overflows. The packet order and the selection of mispredicted packets are generated at random. The assignment rate of generated packets to each of the four WFQ queues is 25 %.

In Table 7.1, the results for a single simulation are shown. It has to be mentioned that the values in the table solely refer to the queuing module. Additional delays that result from protocol processing in the PPEs are not included.

		PQ 1	PQ 2	WFQ 1	WFQ 2	WFQ 3	WFQ 4
PHB				EF	BE	BE	BE
w_i				0.4	0.3	0.2	0.1
WFQ	Mean Delay (cycles)	-	-	600.0	636.3	714.3	953.4
	Delay Jitter (cycles)	-	-	853	858	1341	1907
PQ+WFQ	Mean Delay (cycles)	26.2	44.3	696.6	708.9	758.6	989.4
	Delay Jitter (cycles)	70	93	1017	1187	1503	2031

Table 7.1: Mean delay and jitter

It can be seen that mean delays and the delay jitter for WFQ queues increase with decreasing values for w_i. If mispredicted EF packets are assigned to one of the priority queues, their mean delays can be reduced. The mean delays of WFQ queues however increase if priority queues are applied. In Figure 7.4, the queuing delay distributions of WFQ queue 1 is shown for the reference as well as for the PQ+WFQ system.

The relative frequency of packets is expressed by the relation between queued packets in a queue with a certain delay and the total number of simulation packets in the same queue. Mispredicted EF packets are assigned to priority queues. Consequently, discrete values for packet frequencies in case of pure WFQ are smaller than for PQ+WFQ. According to the results of the table above, an increased delay can be observed for the PQ+WFQ system compared to the WFQ model.

In the following, the targeted delay jitter reduction of the PQ+WFQ model is considered. The jitter reduction takes into account additional processing delays that result from mispredictions and queuing delays. It is calculated as the difference between delay jitter of the

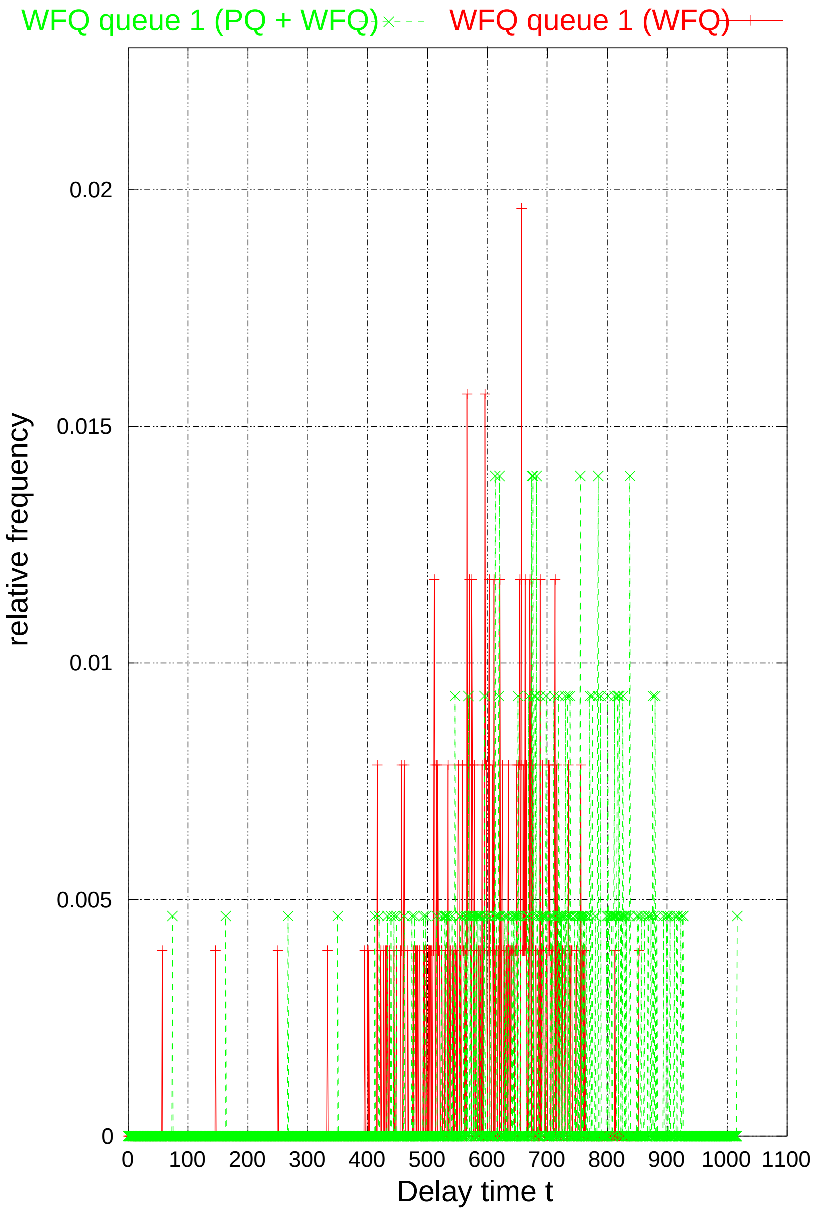


Figure 7.4: Queuing delay EF queue

WFQ and the PQ+WFQ system. The delay jitter j_{PQ+WFQ} of the PQ+WFQ model includes misprediction delays and is:

$$j_{PQ+WFQ} = d_{WFQq1_{max}} - min\{d_{WFQq1_{min}}, d_{PQq1_{min}} + d_{m2}, d_{PQq2_{min}} + d_{m1}\} \tag{7.1}$$

with the maximum delay of WFQ queue 1 $d_{WFQq1_{max}}$, the minimum delay of WFQ queue 1 $d_{WFQq1_{min}}$, the minimum delay in case of 2 mispredictions $d_{PQq1_{min}} + d_{m2}$ and the minimum delay in case of 1 misprediction $d_{PQq2_{min}} + d_{m1}$. The values for d_{m2} and d_{m1} are 600 cycles and 300 cyles, respectively.

The delay jitter j_{WFQ} of the WFQ model including the misprediction delays is:

$$j_{WFQ} = d_{WFQq1_{max}} - d_{WFQq1_{min}} \tag{7.2}$$

with the maximum delay $d_{WFQq1_{max}} + d_x$ and the minimum delay $d_{WFQq1_{min}} + d_y$ of WFQ queue 1. d_x and d_y are addends which consider the misprediction delay. Their values are 0 in case of no, d_{m1} in case of one and d_{m2} in case of two mispredictions.

With equations 7.1 and 7.2 follows for the delay jitter reduction Δ_j

$$\Delta_j = j_{WFQ} - j_{PQ+WFQ} \tag{7.3}$$

The reduction of the delay jitter through priority queues is illustrated in Figure 7.5. The bars represent the range of simulation results for the jitter reduction.

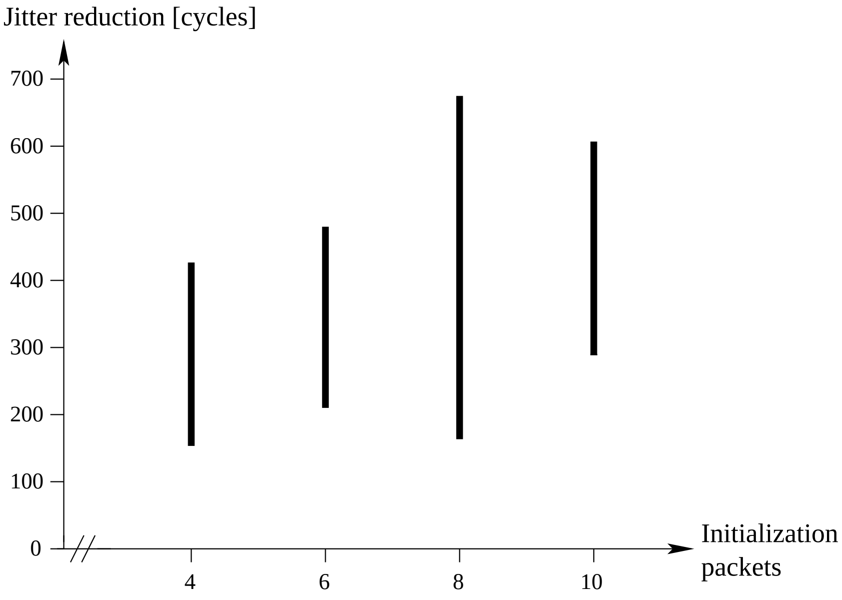


Figure 7.5: Delay jitter reduction

It can be seen that the jitter reduction depends on the number of initialization packets, i.e. the fill level at the beginning of the simulation. A significant deviation for the jitter reduction can be seen as well. This results from the randomized parameters, i.e. the packet order, the packet reception time, the selection of mispredicted packets and the packet assignment to queues. However, a positive value for the delay jitter reduction can be achieved for different numbers of initialization packets. Thus, the delay jitter of the speculative processing model can be compensated by a queuing system that applies additional priority queues. It has to be mentioned that the applied queuing system with priority queues can also be used in a pseudo-parallel model to reduce the queuing delay of EF traffic. In the speculative processing model, however, the queuing system with additional priority queues has been used to cope with the generated delay jitter due to mispredictions.

7.3 Network processor delay

The PPEs and the queuing module have been evaluated in the previous chapters. In order to determine the delay properties of the complete network processor, the receive and transmit units have to be analyzed. It should be noted that this part targets on an estimation of the required processing cycles.

Concerning the receive unit, it is assumed that functions such as HDLC/POS serial-to-parallel conversion, section/path overhead extraction, payload pointer interpretation as well as Ethernet framing, frame size monitoring and FIFO buffering are done in the external POS framer / Ethernet MAC device [111] [150]. An external FIFO is used to buffer incoming data before their transfer to the NP.

The internal receive unit is responsible for dispatching packets, which are kept in the FIFO, to an idle PPE. The data transfer itself is started on request of the PPE. Depending on the data format of the FIFO, reassembly might be required as well as an identification of the port. The packet data is transferred to the input buffer of the assigned PPE. According to [141], 30 cycles are required for a data transfer of 32 bytes from the FIFO to the embedded processor. Consequently, for packets with a minimum size of 96 bytes, 90 cycles are required. Based on [140] and [29] and considering separate packet memories for PPEs, the total processing effort needed for this block is estimated to 180 to 200 compute cycles.

The internal transmit block handles the data movement from the queuing module to a succeeding external device. Enqueued packets that await transmission are kept in a shared DRAM. The requirement for a DRAM read operation, which is derived from [141], is estimated to 55 cycles. Additional information that is furthermore required by a succeeding framer/MAC or a switch fabric has to be appended. In total, the estimated delay is about 125 to 145 cycles [29].

Table 7.2 gives an overview of the processing delay for a complete network processor. The estimated values refer to clock frequencies of the embedded processor cores. Mean PPE delays and the mean delay for EF-packets are taken from Figure 6.13 and Table 7.1. The values are derived from the percentages of none, one and two mispredictions and the corresponding delays. The applied traffic corresponds to the traffic distribution in [101]. The prediction rates are taken from chapter 6.2.

While the pseudo-parallel model requires between 2067 and 2107 cycles, the effort for speculative processing is only between 1757.8 and 1823.8. This corresponds to a latency

Processing model	Speculative	Pseudo-parallel
Queuing algorithm	PQ / WFQ	WFQ
Mean delay PPE [cycl.]	874 - 900	1162
Mean EF-delay [cycl.]	578.8	600
Delay receive unit [cycl.]	180 - 200	180 - 200
Delay transmit unit [cycl.]	125 - 145	125 - 145
$\sum$ [cycl.]	1757.8 - 1823.8	2067 - 2107

Table 7.2: Processing delays

reduction of 13.4 and 14.9 %. If an NP clock of 400 MHz is assumed, mean latencies are observed between the incoming and the outgoing interface amount of 4.4 - 4.6 μs in case of speculative processing and 5.2 - 5.3 μs in case of pseudo-parallel processing, respectively.

7.4 End-to-end analysis

In the previous chapters, the speculative processing model has been evaluated. The obtained latency reduction in a single network node is now compared with the delay of the complete transmission path, i.e. the “end-to-end” delay. As an application example, which possesses end-to-end delay requirements, IP telephony is used.

Network architecture

A common network architecture is depicted in Figure 7.6. Public network domains of ISPs are highlighted as bright clouds.

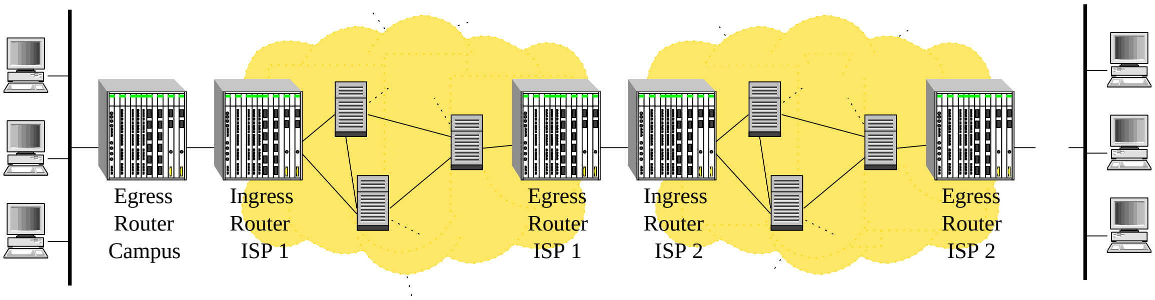


Figure 7.6: Network architecture

End systems of a local-area network (LAN) are connected to a public network infrastructure through a LAN/campus egress router and an ISP ingress router. In case of a dial-in access, a connection to the network is established through a network access router. In both cases, access to a domain is only given if a service-level agreement (SLA) between the end terminal subscriber and the network service provider exists. At this access or edge router, the SLA compliance is verified and, according to the result, packet forwarding, re-classification or dropping is initiated.

According to the results of chapter 4, a significant latency gain for the speculative processing model is anticipated in edge routers where complex protocol processing is done. The number of edge routers along the transmission path can be derived from the number of individual DiffServ domains. According to the SEQUIN initiative of the GEANT research network, premium IP service implementation targets on a minimum number of actions per domain interconnection node [151]. While functionalities such as marking and policing have to be done only in selected nodes, more complex tasks such as admission control, classification, monitoring and accounting have to be performed by every DiffServ edge router, i.e. ingress and egress router. Campus edge routers and firewalls, which might have a demand for deep packet header analysis, are additionally considered. The number of network domain transitions given below in Table 7.3 depends on several factors. The locations of the end systems possess an important role on the number. The domain which is passed first has furthermore an essential impact on the selection of a succeeding ISP domain [152].

The SpaceNet "traceroute" Service and "ping plot" have been taken to estimate the number of network domain transitions for different connection end points [153] [154]. The results are given in Table 7.3. The values for the end-to-end delay reduction are derived from the different processing delays in Table 7.2 and an assumed NP clock of 400 MHz.

Connection end points	No. ISP domain transitions	No. complex processing routers	End-to-End delay reduction (μs)
Munich - Washington, DC	1 - 4	6 - 12	4.2 - 9.6
Munich - San Francisco	1 - 5	6 - 14	4.2 - 11.2
Munich - Sydney	2 - 5	8 - 14	5.6 - 11.2

Table 7.3: Number of complex processing router

It is assumed that each of the passed ISP domains implements DiffServ functionalities. The connection end points refer to different institutions. The end-to-end delay reduction is obtained by the application of the speculative processing model. The complex protocol-processing tasks of the applied routers correspond to the SEQUIN initiative. The number of routers n_{router} can be derived from the number of ISP domain transitions n_{ISP}, the campus ingress and egress router, the ingress router of the first ISP domain and the egress router of the last ISP domain along the transmission path. Thus, it is

$$n_{router} = 2 \cdot n_{ISP} + 2 \tag{7.4}$$

The values for the end-to-end delay reduction refer only to the NP systems. Additional delays that result from network node devices, e.g. framers, MACs and switch fabrics, are not included.

End-to-end delay

In case of IP telephony, the maximum value for an acceptable one-way end-to-end delay is defined to 150 ms [155]. This time budget has to be spread between three parts, namely the transmission source, the network and the transmission sink. The delay portions for these components are shown in Figure 7.7.

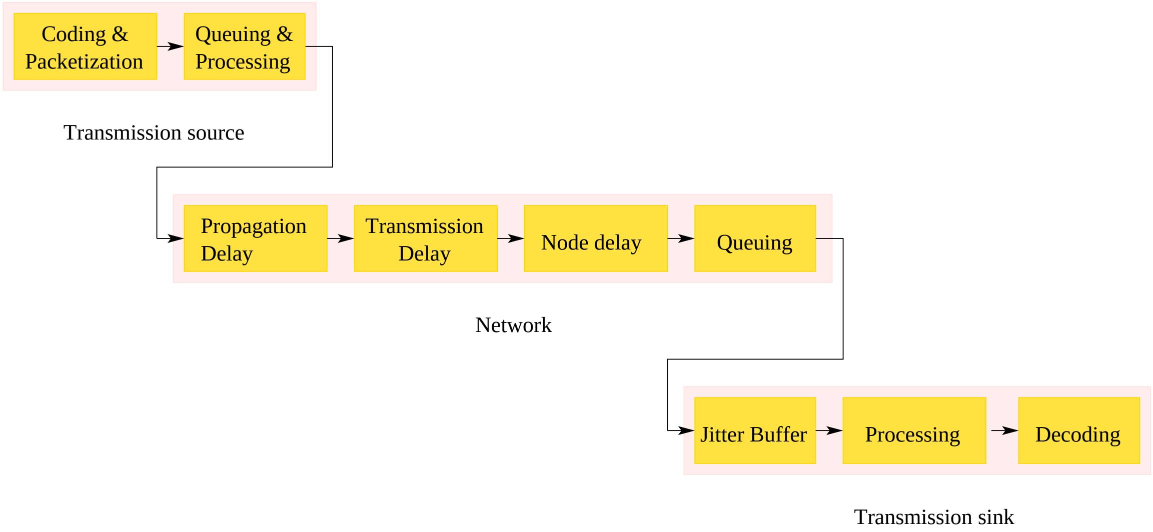


Figure 7.7: Delay portions

A speech encoder in the transmission source converts a digitized signal to a bit-stream, packetizes and sends it over the network. Codecs are therefore applied to code a digital sound stream captured from a sound card and to decode the stream captured from the network to a playable format. Commonly used codecs for IP telephony differ in packet sizes, bit-rates and delays. Some codecs and their delays are G.711 (20 ms), G.729 (25 ms) and G723.1 (37.1 ms) [156]. Additionally, a packet processing and queuing delay of about 20 - 70 ms is generated by the sound card and the operating system [157].

Propagation delays of networks result from transmission medium properties. The propagation velocity in optical fiber is approximatively 0.67 · speed-of-light, i.e. 200000 km/s [25]. The propagation delay for three pairs of end points can be seen in Table 7.4.

Connection end points	Distance [km]	Propagation delay [ms]
Munich - Washington, DC	7214	36.1
Munich - San Francisco	11155	55.8
Munich - Sydney	18932	94.7

Table 7.4: Propagation delay

Transmission delays depend on the data format of the physical medium and the supported data rates. Node delays are composed of packet processing delays in network node devices, e.g. NPs, framers, switch fabrics. Queuing delays result from overloads of components, especially at traffic aggregation points.

A jitter buffer is necessary in the transmission sink to smooth packet burstiness and to compensate delay jitter which has been generated along the transmission path [158]. Typical delay values for jitter buffers range between 30 and 300 ms [157]. The decoder reconstructs the voice signal from the received packets of the stream. The required time for this task is about 1 ms. Finally, the processing time is identical with the processing time in the transmission source, i.e. 20 ms.

Apart from network node delays, common delay values for an end-to-end transmission path between Munich and San Francisco are shown in Table 7.5.

	Delay [ms]	Comment
Coding / Packetization	20	G.711
Queuing / Processing	20 - 70	
Propagation delay	55.8	Munich - San Francisco
Transmission delay	0.016 - 25	100 Mbps / 64 kbps
Queuing	0 - 50	
Jitter buffer	30 - 300	
Processing	20	
Decoding	1	G.711
$\sum$	146.816 - 541.8	

Table 7.5: Delay values

The relative delay reduction obtained by the speculative model along the complete transmission path in relation to the accumulated end-to-end delay is about $20 \cdot 10^{-6}$. If a traffic burst leads to a queuing delay in a single network node, the latency gain obtained by speculative processing has a negligible value. However, an added delay time of more than 150 ms does violate the ITU recommendation [155]. Consequently, a local reduction of the protocol-processing delay in a networking node represents an indispensable contribution for delay sensitive high-speed forwarding.

Chapter 8

Summary and conclusions

An increased performance demand arises for routers in the Internet. In order to cope with this requirement and to provide furthermore the flexibility to support different protocols and services, network processors have been introduced. They provide a high throughput by means of embedded processor cores. Apart from a system throughput, latency is another essential performance metric of networking systems. Short processing latencies are required to enable delay-sensitive applications in a distributed environment. The limited focusing of network processors concerning short protocol-processing latencies motivated this work. The main topic of this thesis is a speculative protocol-processing methodology for network processors that is focused on short processing latencies. A prediction is applied to predict the protocol-stack of packets that will be received next. This knowledge is used to overcome data dependencies in the packet header and to process protocol tasks of the packet synchronously in parallel processing units. The targeted benefit is a reduced mean processing latency compared to traditional processing methodologies.

A concept evaluation was done to determine the latency reduction that can be obtained by the speculative processing concept. The analysis has been performed under consideration of parameters that have an impact on the latency and the workload of the system. The results exhibit that the latency can be significantly reduced, if the prediction covers multiple layers. Thus, a promising application field for the speculative processing concept is where complex processing is required, e.g. in edge routers or gateways. The contrary appears for the speculative concept if processing in different layers is limited, e.g. IPSec or VPN. While encryption and decryption are done in the transmission end points, no deep packet header analysis has to be performed along the transmission path. The sequence of processing tasks for the applied protocol-stack obeys logical considerations, e.g. an address lookup is only performed if the checksum calculation of the packet header was correct. The results of the concept evaluation however exhibit that the sequence and the position of the processing task that derives the type of the upper protocol layer has a considerable impact on the delay. Thus, it can be concluded that an optimization of the task order has to be done to achieve a maximum value for the delay reduction. The analysis showed that an optimum value for the prediction accuracy leads to a substantial delay reduction of up to 28 %, while the additional processing workload decreases to 0 %. Consequently, a high value for the prediction accuracy has to be striven.

Leaving behind these general insights, a transition is made to the Internet and to VLSI-specific considerations. A network processor architecture is therefore specified, which offers multiprocessing capabilities. The protocol-processing units, which represent the so-called fast-path, are realized as multiple protocol-processing elements. Each of them is built up of four embedded processors. The implemented functionalities comprise tasks of an edge router in a DiffServ domain. The protocol-stack prediction unit implements an algorithm which is based on the principle of most-frequently used events. Simulations of the implemented dynamic prediction model have been performed with current network traffic distributions. It has been demonstrated that the algorithm possesses the capability for a smooth adaptation to altered prediction values. Based on current traffic distributions that have an dominant percentage of TCP, values for the prediction accuracy range of up to 82 to 90 % for the first prediction stage, and up to 98 to 99 % for the first and second prediction stage. If the percentage of UDP traffic increases, e.g. for RT-applications, the prediction hit rate will be decreased. It has been additionally shown that the algorithm, which is capable for an update of the prediction registers in packet-rate, can be implemented in software. However, the packet data collection at the input of the prediction computation unit has to be enhanced to provide updated prediction values for network processor systems with multiple packet-processing elements. Weight factors have been introduced to the prediction unit to priorize protocol-processing of delay-sensitive packets even if their reception frequencies do not correspond to the most-frequently used protocol-stacks. In order to obtain maximum values for the latency reduction under consideration of prioritized packets and their reception frequencies, a necessity for an exact assignment of weight factor values arises. The large number of special registers and the computation effort for the calculation of the required MFU values show weaknesses of the dynamic prediction algorithm. An alternative algorithm with less complexity has to be striven, e.g. a tree search algorithm. These topics will be subjects of further research.

The main intention of the system design was not to derive an optimized architecture of a speculative network processor. Instead, an exemplary platform design was defined for evaluation purposes. Based on the same amount of processor resources, the speculative and a pseudo-parallel reference processing model have been mapped to this architecture. Performed simulations observed the relationship of both models in terms of processing delay and processor workload. The scenario which was used for the system evaluation was a forwarding device in an ISP edge router. Based on this, it was found that the latency of the speculative model can be significantly decreased to a value of 6.4 to 22.5 % compared to the reference model.

However, functionalities which are time-consuming, e.g. DiffServ tasks such as multifield-classification and policing, lead to a high workload of the assigned processors. Since occurring mispredictions require an additional processing effort, an essential increase of the processor workload is given. Another consequence is the fact that the workload of the busiest NL-PUs determines the processing delay. This requires a balanced workload of the NL-PUs. Further research is necessary to develop an improved model for the assignment of processing tasks to NL-PUs. The aim of the model is a balanced workload to achieve minimum values for the processing delay. Parameters which seem to be relevant for this study are network traffic distributions, the corresponding protocol-stack prediction and the applied protocol-stack.

A simultaneous initiation of layer-processing in different processors additionally involves a higher number of simultaneous access requests on the local SRAM. The performed simulations consequently have shown that the bus arbiter represents a performance bottleneck of the speculative system. The shared bus workload is increased as well. In order to decrease the number of simultaneous access requests and to decrease the arbiter load, a conceivable approach might be to modify the PPE architecture in terms of the memory assignment to NL-PUs. In the applied architecture, each PPE contains one local SRAM to which which three NL-PUs have access. An alternative approach is to assign multiple NL-PUs of different packets and different layers to one SRAM. Consequently, different packet reception times enable that times of input, protocol-stack and output processing are distributed and the number of simultaneous SRAM access requests would be decreased. Another alternative to decrease the bus workload that mainly results from DiffServ tasks such as multifield classification might be to shift these tasks to acceleration units which are external to PPEs.

In the implemented model, the bus access is statically prioritized for the NL-PUs from layer 2 to 4. A further improvement of the processing delay might be possible if the shared bus is granted in an alternative way, e.g. round-robin. This can be subject of further research.

As a prerequisite for the latency gain, sufficient processing resources are required to avoid stall times in case of successively mispredicted packets. The amount of processing resources has to be predefined before the implementation of the system. If this value is derived from the amount of resources that are required for processing mispredicted packets with two mispredictions, the system is overprovisioned if the prediction hit rate increases. The pseudo-parallel reference model has a higher processing latency. Thus, due to a lack of mispredictions and a reduced workload, the throughput of a reference system that possesses an identical amount of processing resources is higher than the throughput of a speculative one. This implies that unused processor workload can be assigned to additionally dispatched packets.

In order to develop an optimized system architecture, an effort has to be spent to analyze if a balanced workload and consequently a reduced latency can be achieved by architectural improvements. A conceivable starting point for this might be the replacement of scalar NL-PUs through multithreading processor cores.

As a consequence of correctly predicted and mispredicted packets deviations appear for the protocol-processing latency. In order to compensate this delay jitter, an approach has been presented which can be applied to the speculative network processor model. The applied queuing algorithm is therefore extended by two priority queues, which mispredicted EF-packets are assigned to. The performed simulations clearly demonstrated that the delay jitter which has been generated in packet processing elements can be compensated and that the speculative model is suitable for delay-sensitive traffic.

The proposed concept contains a prediction that provides not only a statement about the types of protocol-layers. It also indicates the control-flow path through the protocol-stack, e.g. whether the packets have to be forwarded or terminated. It furthermore predicts the presence of optional protocol-header fields in a packet and makes an assumption whether packet header data are corrupted or not. In the exemplarily implemented NP architecture, protocol-processing has been done in three NL-PUs. Further research activities have to been spent to scale the speculative processing model to more than three processing units. This requires on one hand a model for an efficient assignment of protocol tasks to embedded processors. On the other hand, considerations about the predicted statements towards value prediction have to be done, e.g. prediction of CoS-labels.

On the whole, it can be concluded that the proposed methodology of protocol-stack prediction and speculative packet-processing represents an essential contribution for reduced processing delays in network processors. It can be profitably applied to network-edge routers and domain border routers.

Bibliography

[1] B. Leiner, et al., *A Brief History of the Internet* Internet Society, www.isoc.org., Reston, Version 3.31, Aug. 2000.

[2] D. Comer, *Internetworking with TCP/IP* Volume 1, 3rd edition, Prentice-Hall, Upper Saddle River, 1995.

[3] *ISO 7498. Information Processing Systems - Open Systems Interconnection - Basic Reference Model* International Standard, ISO, 1984.

[4] U. Black, *OSI A Model for Computer Communications Standards* Prentice Hall, Eaglewood Cliffs, 1991.

[5] W. Haass, *Handbuch der Kommunikationsnetze, Einführung in die Grundlagen und Methoden der Kommunikationsnetze* Springer Verlag Berlin, 1997.

[6] A. Tanenbaum, *Computer-Networks* Second edition, Prentice-Hall, Eaglewood Cliffs, 1989.

[7] S. Giordano, S. Salsano, S. Van der Berghe, G. Ventre, D. Giannakopoulos, *Advanced QoS Provioning in IP Networks: The European Premium IP Projects* IEEE Communications Magazine, Vol 41 No.1, Jan. 2003.

[8] R. Braden et al., *Integrated Services in the Internet Architecture: an Overview* IETF Request for Comment 1633, Jun. 1994.

[9] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, *An architecture for differentiated services* IETF, RFC 2475, Dec. 1998.

[10] K. Kilkki, *Differentiated Services for the Internet* Macmillan Technical Publishing, Indianapolis, 1999.

[11] K. Nichols, S. Blake, F. Baker, D. Black, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers* IETF, RFC 2474, Dec. 1998.

[12] P. Pradhan, T. Chiueh, *A Cluster-based Scalable and Extensible Edge Router Architecture* ECSL Technical Report 2000-79, DCS, State University of New York, 2000.

[13] E. Rosen, A. Viswanathan, R. Callon, *Multiprotocol Label Switching Architecture* IETF, RFC 3031, Jan. 2001.

[14] B. Gleeson et al. *A framework for IP Based Virtual Private Networks* IETF, RFC 2764, Feb. 2000.

[15] L. Kleinrock, *Queueing Systems, Volume 2: Computer Applications* John Wiley & Sons, New York, 1976.

[16] K. Jeffay, *Rate-based execution models for real-time multimedia computing* Dept. of Computer Science, University of North Carolina at Chapel, Sep. 1997

[17] L. Kleinrock, *Queueing Systems, Volume 1: Theory* John Wiley & Sons, New York, 1975.

[18] J. Sterbenz, *Protocol for High-Speed Networks: A Brief Retrospective Survey of High-Speed Networking Research* Proc. of Protocol for High Speed Networks, Apr. 2002.

[19] W. Bux, W. Denzel, T. Engbersen, A. Herkersdorf, R. Luijten *Technologies and Building Blocks for Fast Packet Forwarding* IEEE Communications Magazine, Jan. 2001.

[20] L. Gwennap, B. Wheeler, *A Guide to Network Processors* Published by MicroDesign Resources, Sunnyvale, 2000.

[21] *Intel IXP1200 Network Processor* Advance Datasheet, Intel, Sep. 1999.

[22] *IBM Network Processor Databook* IBM Microelectronics Division, Nov. 1999.

[23] *Intel IXP1200 Network Processor: Software Reference Manual* Level One Communnications, Inc., an Intel company, Mar. 2000.

[24] S. Hily, A. Seznec, *Branch Prediction and Simultaneous Multithreading* Pulication interne No. 997, IRISA, Rennes, 1996.

[25] J.P.G. Sterbenz, J.D. Touch, *High-Speed Networking: A Systematic Approach to High-Bandwidth Low-Latency Communication* John Wiley, New York USA, 2001.

[26] P. Paulin, F. Karim, P. Bromley, *Network Processors: A Perspective on Market Requirements, Processor Architectures and Embedded S/W Tools* Proc. of DATA 2001, Munich, Mar. 2001.

[27] M. Heddes, E. Ruetsche, *A Survey of Parallelism in Communication Subsystems* IBM Zurich Research Laboratory, Res. Rep. RZ 2570, 1994.

[28] G. Federkow, *Technology Overview* Technology of Edge Aggregation: Cisco 10000 Series Edge Services Router, Cisco Systems, Inc. 2003.

[29] S. Lakshmanamurthy, K. Liu, Y. Pun, L. Huston, U. Naik, *Network Processor Performance Analysis Methodology* Intel Technical Journal, Vol. 6, Aug. 2002.

[30] F. Baker, *Requirements for IP version 4 routers* RFC 1812, IETF, Jun. 1995.

[31] Y. Rekhter, T. Li, *An architecture for IP address allocation with CIDR* RFC 1518, IETF, Sep. 1993.

[32] P. Gupta, *Routing lookups and packet classification* Tutorial at AT&T Research, Florham Park, Jul. 2000.

[33] A. Demers, S. Keshav, S. Shenker, *Analysis and Simulation of a fair queueing algorithm* Internet Res. and Exper., vol. 1, 1990.

[34] A. Parekh, R. Gallager, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case* IEEE/ACM Transactions on Networking, Vol. 1, No. 3, Jun. 1993.

[35] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet and the Telephone Network* Professional Computing Series, Addison Wesley, New York, 1997.

[36] *Quality of Service Solutions Configuration Guide* Cisco IOS Software Releases 12.0, 2002.

[37] V. Cerf, R. Kahn, *A Protocol for Packet Networking Interconnection* IEEE Transactions on Communications Technology, vol. comm-22, no. 5, New York, May 1994.

[38] J. Postel, *User Datagramm protocol* RFC 768, STD 006, Aug. 1980.

[39] T. Braun, B. Stiller, M. Zitterbart, *OSI or Special Protocols for High Speed Networks ?* Proc. 2nd joint Workshop on High-Speed Networks, 1991.

[40] T. Braun, M. Zitterbart, *Parallel Transport System Design* IFIP Conference on High-Performance Networking, Liege, 1992.

[41] G. Chesson, *XTP/PE Design Considerations* IFIP PfHSN 1989, Zurich, May 1989.

[42] D. Cheriton, *VMTP: A Transport Protocol for the Next Generation of Computer Systems* Proc. of ACM SIGCOMM 1986, Computer Communication Review, vol.16, no.3, ACM, New York, Aug 1986.

[43] M. Zitterbart, B. Stiller, A. Tantawy, *A Model for Flexible High-Performance Communication Subsystems* IEEE Journal on Selected Areas in Communications 11, May 1993.

[44] T. Plagemann, B. Plattner, M. Vogt, T. Walter, *A Model for Dynamic Configuration of Light-Weight Protocols* 3rd workshop on Future Trends of Distributed Computing Systems, Taipeh, Apr. 1992.

[45] Z. Haas, *A Protocol Structure for High-Speed Communications over Broadband ISDN* IEEE Network, Jan. 1991.

[46] D. Box, D. Schmidt, T. Suda, *Adaptive - An Object Oriented Framework for Flexible and Adaptive Communication Protocols* 4th IFIP Conference on High Performance Networking, Liege, Dec. 1992.

[47] E. Nahum, D.J. Yates, S.O'Malley, H. Orman, R. Schroeppel, *Parallelized Network Security Protocols* ISOC Symposium on Network and Distributed System Security, 1996.

[48] M. Bjoerkman, P. Gunningberg, *Locking Effects in Multiprocessor Implementations of Protocols* Proc. ACM Sigcomm 93, Sep. 1993.

[49] D. Patterson, J. Hennessy, *Computer Organization and Design: The Hardware/Software Interface* Morgan Kaufmann Publishers, 1994.

[50] *MAE East Network Access Point* www.mae.net

[51] D. Clark, V. Jacobson, J. Romkey, H. Salwen, *An Analysis of TCP processing overhead* IEEE Communications Magazine, 27(6), Jun. 1989.

[52] D. Clark, D. Tennenhouse, *Architectural Considerations for a New Generation of Protocols* Proc. SIGCOMM'90, pages 200-208, Sept. 1990.

[53] M. Abbott, L. Peterson, *Increasing Network Throughput by Integrating Protocol Layers* ACM Transactions on Networking, vol. 1, issue 5, Oct. 1993.

[54] B. Ahlgren, M. Bjoerkman, P. Gunningberg, *The Applicability of Integrated Layer Processing* 7th IFIP Conference on High Performance Networking, White Plains, Apr. 1997.

[55] W. R. Stevens, *TCP/IP Illustrated, Volume 1* Addison-Wesley, Logman Inc., 1999.

[56] V. Jacobson, *Congestion Avoidance and Control* ACM Computer Communication Review, Oct. 1988.

[57] V. Jacobson, *4BSD TCP Header Prediction* Computer Communication Review, vol. 20, no. 2, Apr. 1990.

[58] C.M. Woodside, K. Ravinadran, R.G. Franks, *The Protocol Bypass Concept for High Speed OSI Data Transfer* IFIP Workshop on Protocols for High Speed Networks, Nov. 1990.

[59] C. Smith, *Independent General Principles for Constructing Responsive Software Systems* ACM Trans. on Computer Systems, vol. 4, Feb. 1986.

[60] K. Ravindran, G. Singh, C. Woodside, *Architectural Concepts in Implementation of End-system Protocols for High Performance Communications* IEEE International Conference on Network Protocols, Oct. 1996.

[61] D. Giarizzo, M. Kaiserswerth, T. Wicki, R. Williamson, *High-Speed Parallel Protocol Implementations* Proc. 1st Workshop on High-Speed Networks, May 1989.

[62] D.C. Schmidt, T. Suda, *The Performance of Alternative Threading Architectures for Parallel Communication Subsystems* Journal of Parallel and Distributed Processing, 1996.

[63] D.C. Schmidt, T. Suda, *Adaptive: A Framework for Experimenting with High-Performance Transport System Process Architectures* International Conference on Computer Communication Networks, San Diego, Jun. 1993.

[64] S. Matheson, *Steering your Way through Net Processor Architectures* Silicon Access Networks, www.commsdesign.com, Jul. 2002.

[65] M. Hamdan, *A Combinatorial Framework for Parallel Programming using algorithmic Sceletons* Thesis, Dept. Comptuing and Electrical Engineering, Edinburgh, Jan. 2000.

[66] J. Hennessy, D. Patterson, *Computer Architecture, A Quantitative Approach* 2nd Edition, Morgan Kaufmann Publishers, Inc., San Francisco, 1996.

[67] M. Kaiserswerth, *The Parallel Protocol Engine* IEEE/ACM Transactions on Networking, 1993.

[68] K. Torp, *Object-oriented programming* Presentation, Dept. of Computer Science, Aalborg University, Sep. 2002.

[69] A. Schuster, *Concurrent and Distributed Programming* CS236370, Technion - Israel Institute of Technology, 1998.

[70] J. Lo, *Exploiting Thread-Level Paralleleism on Simultaneous Multithreaded Processors* Thesis, University of Washington, 1998.

[71] S. Storino, A. Aipperspach, J. Borkenhagen, S. Levenstein, *A commercial multithreaded RISC processor* International Solid State Circuits Conference, Feb. 1998.

[72] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Poerterfield, B. Smith, *The Tera computer system* International Conference on Supercomputing, Jun. 1990.

[73] D. Tullsen, S. Eggers, H. Levy, *Simultaneous Multithreading: Maximizing On-Chip Parallelism* Proc. of 22nd Annual International Symposium on Computer Architecture, Santa Kargherita Ligure, Jun. 1995.

[74] J.-M. Parcerisa, A. Gonzalez, *Improving Latency Tolerance of Multithreading through Decoupling* IEEE Transactions on Computers, vol. 50, Oct. 2001.

[75] J. Foag, T. Langguth, M. Leibiger, T. Messerer, N. Pazos, T. Wild, *Programmierbarer Parser-Prozessor - Untersuchungen zu Netzprozessoren* Project Report LIS FhG-ESK Siemens AG, Jan. 2001.

[76] S. Wallace, B. Clader, D. Tullsen, *Threaded Multiple Path Execution* 25th International Symposium on Computer Architecture, Jun. 1998.

[77] K. Wang, M. Franklin, *Highly Accurate Data Value Prediction using Hybrid Predictors* 30th Annual Symposium on Microarchitecture Micro'97, Dec. 1997.

[78] F. Thiesing, *Analyse und Prognose von Zeitreihen mit Neuronalen Netzen* Shaker Verlag, Aachen, 1998.

[79] T. Yeh, D. Long, S. Brandt, *Increasing Predictive Accuracy by Prefetching Multiple Program and User Specific Files* Proc. of Intern. Symposium for High Performance Computing Systems and Applications (HPCS) 2002, Moncton, Jun. 2002.

[80] A. Afsahi, N. Dimopoulos, *Architectural Extensions to Support Efficient Communication Using Message Prediction* Proc. of Intern. Symposium for High Performance Computing Systems and Applications (HPCS) 2002, Moncton, Jun. 2002.

[81] Z. Su, *A Comparative Analysis of Branch Prediction Schemes* Lesson, Computer Science Division, Univ. of California, Berkeley, 1995.

[82] S. McFarling, *Combining Branch Predictors* WRL, Technical Note TN-36, Digital, Western Research Laboatory, Palo Alto, Jun. 1993.

[83] B. Calder, D. Grunwald, *Fast & Accurate Fetch and Branch Prediction* Intern. Symposium on Computer Architecture, Chicago, Mar. 1994.

[84] D. Patterson, J. Hennessy, *Computer Architecture: A Quantitative Approach* 2nd edition, Morgan Kaufmann Publishers, Inc. 1995.

[85] T. Yeh, Y. Patt, *Alternative Implementations of Two-level Adaptive Branch Prediction* Intern. Symposium on Computer Architecture, Gold Coast, May 1992.

[86] M. Evers, S. Patel, R. Chappell, Y. Patt, *An Analysis of Correlation and Predictability: What makes Two-Level Branch Predictors Work* Intern. Symposium on Computer Architecture, Barelona, Jun. 1998.

[87] B. Rychlik, J. Faistl, B. Krug, J. Shen, *Efficacy and Performance Impact of Value Prediction* Intern. Conf. on Parallel Architectures and Compilation Techniques, Paris, Oct. 1998.

[88] L. Trajkovic, A. Neidhardt, *Traffic characterization and time scales for designing efficient network control policies* Invited paper, Proc. NOLTA'97, Hawaii, Dec. 1997.

[89] L. Trajkovic, A. Neidhardt, *Internet traffic prediction* Centre for Systems Science, Simon Fraser University, Vol. 12, Issue 1, Mar. 2000.

[90] M.E. Crovella, A. Bestavros, *Self-similarity in world wide web traffic: Evidence and possible causes* IEEE/ACM Transactions on Networking, vol. 6, pp. 835-846, Dec. 1997.

[91] W.E. Leland, M.S. Taqqu, W. Willinger, D.V. Wilson, *On the self-similar nature of ethernet traffic* IEEE/ACM Transactions on Networking, vol. 2, pp. 1-15, Feb. 1994.

[92] A. Feldmann, A. Gilbert, W. Willinger, T. Kurtz, *The Changing Nature of Network Traffic: Scaling Phenomena* ACM Computer Communication Review, Apr. 1998.

[93] L. Guo, M. Crovella, I. Matta, *TCP Congestion Control and Heavy Tails* Technical Report BU-CS-2000-017, Boston University, Computer Science Department, Jul. 2000.

[94] J. Beran, *Statistics for Long-Memory Processes* Monographs on Statistics and Applied Propability, Chapman and Hall, New York, 1994.

[95] V. Paxson, S. Floyd, *Wide-Area Traffic: The Failure of Poisson Modeling* IEEE/ACM Transactions on Networking, Jun. 1995.

[96] W. Willinger, M. Taqqu, R. Sherman, D. Wilson, *Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level* IEEE/ACM Transactions on Networking, Feb. 1997.

[97] A. Veres, B. Hungary,, M. Boda, *The Chaotic Nature of TCP Congestion Control* Proc. of IEEE INFOCOM 2000, Mar. 2000.

[98] V. Paxson, *End-to-End Routing Behavior in the Internet* IEEE/ACM Transactions on Networking, Oct. 1997.

[99] K. Claffy, *Internet Measurement and Data Analysis: Topology, Workload, Performance and Routing Statistics* NAE Workshop, 1999.

[100] K. Thompson, G. Miller, R. Wilder, *Wide-Area Internet Traffic Patterns and Characteristics* IEEE Network, 11(6), Nov. 1997.

[101] C. Fraleigh, S. Moon, C. Diot, B. Lyles, F. Tobagi, *Packet-Level Traffic Measurement from a Tier-1 IP Backbone* Technical Report TR01-ATL110101, Sprint ATL Technical Report, Feb. 2002.

[102] S. Bhattacharyya, C. Diot, J. Jetcheva, N. Taft, *Pop-Level and Access-Link-Level Traffic Dynamics in a Tier-1 POP* ACM Sigcomm Internet Measurement Workshop IMW 2001, San Francisco, Nov. 2001.

[103] J. Pino, B. Singh, D. Culler, *Performance Evaluation of One and Two-Level Dynamic Branch Prediction Schemes over Comparable Hardware Costs* Technical Report, UCB/ERL M94/45, Jun. 1994.

[104] J. Smith, *A Study of Branch Prediction Strategies* Proc. 8th Intern. Symposium of Computer Architecture, May 1981.

[105] P. Srisuresh, *Security Model with Tunnel-mode IPSec for NAT Domains* IETF, RFC 2709, Oct. 1999.

[106] *Cisco 10000 Series Internet Routers* Technical manual, Cisco, 2002.

[107] *St200 Service Edge Router* Data Sheet, Laurel Networks, 2002.

[108] *Spring Tide 7000 IP Service Switch Router* Data Sheet Brochure, Lucent technologies, 2002.

[109] *Standard IEEE 802.3z* IEEE Gigabit Task Force, Jul. 1998.

[110] W. Simpson, *PPP in HDLC-like Framing* IETF, Request for Comments, Jul. 1994.

[111] *Khatanga 10 Gigabit Ethernet MAC and PHY/OC-192c POS Framer and Mapper* Product Brief Version 1.1, Apr. 2001.

[112] *PHY/Framer ermoeglichen die Konvergenz von Protokoll-Services* Telecomm & Elektronik, Pages 26-28, Feb. 2001.

[113] P. Gupta, N. Mc Keown, *Packet classification on multiple fields* Proc. ACM SIGCOMM 99, Cambridge, Aug. 1999.

[114] S. Shenker, C. Partridge, R. Guerin, *Specification of Guaranteed Quality of Service* IETF Request for Comment 2212, Sep. 1997.

[115] *Programmierbarer Parser-Prozessor* Projekt-Abschlussbericht, Technische Universitaet Muenchen, Lehrstuhl fuer Integrierte Schaltungen, Jan. 2000.

[116] W. R. Stevens, *TCP/IP Illustrated, Volume 2* Addison-Wesley, Logman Inc., 1999.

[117] *KAME Project: free IPv6 and IPSec-Stack for BSD* www.kame.net, Japan, 2002.

[118] P. Eles, A. Doboli, P. Pop, Z. Peng *Scheduling with Bus Access Optimization for Distributed Embedded Systems* Proceedings of Design, Automation and Test in Europe 1998, pp132-138.

[119] C. Hoare, *Quicksort* Comput. J.5, 1962.

[120] R. Sedgewick, *Algorithms* 2nd edition, Addison Wesley, Reading, MA, 1992.

[121] M. Zitterbart, *Parallelism in Communication Subsystems* in: A. Tantawy: High Performance Communications, Kluwer Academic Publisher, 1993.

[122] D. Yates, *Connection-level Parallelism For Network Protocols on Shared-Memory Multiprocessor Servers* Thesis, directed by J. Kurose, University of Massachusetts Amherst, 1997.

[123] T. La Porta, M. Schwartz, *A High-Speed Protocol Parallel Implementation: Design and Analysis* in: A. Danthine, O. Spaniol: High Performance Networking, IV, IFIP, North-Holland 1993.

[124] T. Braun, C. Schmidt, *A Parallel Transport Subsystem Implementation for High Performance Communication* Concurrency: Practice and Experiment, Special Issue on High Performance Distributed Computing, Jun. 1994.

[125] P. Crowley, M. Fiuczynski, J. Baer, B. Bershad, *Characterizing Processor Architectures for Programmable Network Interfaces* Proc. International Conference on Supercomputing, Santa Fe, May. 2000.

[126] J. Allen, B. Bass, C. Basso, R. Boivie, J. Calvignac, G. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. Logan, M. Peyravian, M. Rinaldi, R. Sabhikhi, M. Siegel, M. Waldvogel, *PowerNP Network Processor: Hardware, Software And Applications* IBM Journal of Research and Development, 2002.

[127] *IX Architecture Whitepaper* LevelOne, An Intel Company, 1999.

[128] *C-5 DCP Architecture Guide* Version 1.1, C-Port Corp., N. Andover, Nov. 1999.

[129] *IXP 1200 Network Processor Hardware Reference Manual* Level One, An Intel Company, 2000.

[130] *Enterprise Systems Architecture /390, Principles of Operation* IBM, SA22-7201-04, Fifth Edition, Poughkeepsie, Jun. 1997.

[131] K. Claffy, G. Miller, K. Thompson, *The Nature of the Beast: Recent traffic measurements from an Internet Backbone* Proc. of ISOC INET'98, Jul. 1998.

[132] P. Sagmeister, G. Dittmann, A. Herkersdorf, D. Webb, *Methodology for Testing High-Speed Network Devices with Predicted Traffic* Proc. of Gigabit Networking Workshop GBN01, Anchorage, Apr. 2001.

[133] A. Gerstlauer, R. Doemer, J. Peng, D. Gajski, *System Design, A Practical Guide with SpecC* Kluwer Academic Publishers, Boston, 2001.

[134] D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer, S. Zhao, *SpecC: Specification Language and Methodology* Kluwer Academic Publishers, Boston, 2000.

[135] *http://www.specc.org*

[136] T. Groetker, S. Liao, G. Martin, S. Swan, *System Design with SystemC* Kluwer Academic Publishers, Boston, 2002.

[137] *http://www.systemc.org*

[138] T. Groetker, Stan Liao, Grant Martin, Stuart Swan, *System Design with SystemC* Kluwer Academic Publishers, Boston, 2002.

[139] J. Gerlach, W. Rosenstiel, *System Level Design Using the SystemC Modeling Platform* 2000.

[140] R. Balan, U. Hengartner, *Performance Analysis of the Intel IXP1200 Network Processor* Project Report for Graduate Architecture Class, Carnegie Mellon University, Sep. 2000.

[141] T. Spalink, S. Karlin, L. Peterson, *Evaluating Network Processors in IP Forwarding* Technical Report TR-626-00, Dept. of Comp. Science, Princeton University, Jan. 01.

[142] *Intel IXP 425 Network Processor* Product Brief, Intel Corp., 2003.

[143] C. Aras, J.F. Kurose, D.S. Reeves, H. Schulzrinne, *Real-time Communication in Packet-Switched Networks* Proc. of IEEE, Vol. 82 (1), Jan. 1994.

[144] D. Verma, H. Zhang, D. Ferrari, *Delay Jitter Control for Real-Time Communication in a packet Switching Network* Proc. of TriComm 91, Chapel Hill, Apr. 1991.

[145] A. Bashandy, E. Chong, A. Ghafoor, *Network Modeling and Jitter Control for Multimedia Communication over Broadband Network* Proc. IEEE Infocom 1999, New York, Mar. 1999.

[146] *Network-on-a-chip solutions from Vitesse* Product Brief, Vitesse Semiconductor Corp., Jan. 2000.

[147] K. Law, *The Bandwidth Guaranteed Prioritized Queuing and Its Implementations* IEEE Globecom 97, Phoenix, Nov. 1997.

[148] T. Ferrari, G. Pau, C. Raffaelli, *Measurement Based Analysis of Delay in Priority Queuing* Internet Performance Symposium, San Antonio, Nov. 2001.

[149] H. De Meer, H. Knoche, *Quality of Service Parameters: A Comparative Study* Technical Report, University of Hamburg, 1997.

[150] *Intel IXF6048/IXF6012 Theory of Operation* Intel, 2002.

[151] R. Sabatino, M. Campanella, *SEQUIN: QoS for GEANT* 2nd Intl. Workshop on Quality of Future Internet Services QofIS 2001, Coimbra, Sep. 2001.

[152] G. Fankhauser, D. Schweikert, B. Plattner, *Service Level Agreement Trading for the Differentiated Services Architectures* Swiss Federal Institute of Technology, Computer Engineering and Networks Lab, Technical Report No. 59, Nov. 1999.

[153] *Pingplotter* Version 1.10, www.pingplotter.com.

[154] *SpaceNet Traceroute Service* www.space.net.

[155] *One-Way Transmission Time* International Telecommunications Union, ITU-T Recommendation G.114, Feb. 1996.

[156] *Speech Coding and Speech Quality in IP Telephony* Global IP Sound, Inc., 2001.

[157] B. Hammer, *IP-Telephony Quality of Service* Vortrag an Universität Klagenfurt, Siemens AG, www-itec.uni-kul.ac.at, May 2001.

[158] D. Lin, *Real-Time Voice Transmissions over the Internet* Master of Science Thesis, University of Illinois, 1999.

Appendix A

Publications and Patent Applications

Publications

- J. Foag et al.,
  Processing methodology and architecture for high-speed networking using protocol-stack prediction 7th International IFIP/IEEE Workshop on Protocols for High-Speed Networks , Berlin, Apr. 2002.

- J. Foag et al.,
  Self-adaptive Parallel Processing Architecture for High-Speed Networking Proc. Int'l. IEEE Symposium on High Performance Computing Systems and Applications, Moncton, Jun. 2002.

- J. Foag et al.,
  Predictive Methodology For High-Performance Networking Proc. 7th Int'l. IEEE Symposium on Computers and Communications, Giardini Naxos, Jul. 2002.

- J. Foag et al.,
  Performance Evaluation of a Speculative Network Processor Proc. Int'l. Symposium on High Performance Computing Systems and Applications, Sherbrooke, May 2003.

- J. Foag and T. Wild,
  Traffic Prediction Algorithm for Speculative Network Processors Proc. Int'l. Symposium on High Performance Computing Systems and Applications, Sherbrooke, May 2003.

- J. Foag and T. Wild,
  Queuing Algorithm for Speculative Network Processors Proc. Int'l. Symposium on High Performance Computing Systems and Applications, Winnipeg, May 2004.

Patent Applications

J. Foag and T. Wild,
Verfahren und Vorrichtung zum Verarbeiten von Datenblöcken
German Patent DE10115799A1, Oct. 2002.