Scalable Application- and User-aware Resource Allocation in Enterprise Networks Using End-Host Pacing

CHRISTIAN SIEBER, Chair of Communication Networks, Technical University of Munich, Germany
SUSANNA SCHWARZMANN, FG INET, TU Berlin, Germany
ANDREAS BLENK, Chair of Communication Networks, Technical University of Munich, Germany and Faculty of Computer Science, University of Vienna, Austria
THOMAS ZINNER, FG INET, TU Berlin, Germany
WOLFGANG KELLERER, Chair of Communication Networks, Technical University of Munich, Germany

Providing scalable user- and application-aware resource allocation for heterogeneous applications sharing an enterprise network is still an unresolved problem. The main challenges are: (i) How to define user- and application-aware shares of resources? (ii) How to determine an allocation of shares of network resources to applications? (iii) How to allocate the shares per application in heterogeneous networks at scale? In this paper we propose solutions to the three challenges and introduce a system design for enterprise deployment.

Defining the necessary resource shares per application is hard, as the intended use case, the user's environment, e.g., big or small display, and the user's preferences influence the resource demand. We tackle the challenge by associating application flows with utility functions from subjective user experience models, selected Key Performance Indicators, and measurements. The specific utility functions then enable a mapping of network resources in terms of throughput and latency budget to a common user-level utility scale. A sensible distribution of the resources is determined by formulating a multi-objective mixed integer linear program to solve the throughput- and delay-aware embedding of each utility function in the network for a max-min fairness criteria. The allocation of resources in traditional networks with policing and scheduling cannot distinguish large numbers of classes and interacts badly with congestion control algorithms. We propose a resource allocation system design for enterprise networks based on Software-Defined Networking principles to achieve delay-constrained routing in the network and application pacing at the end-hosts.

The system design is evaluated against best effort networks in a proof-of-concept set-up for scenarios with increasing number of parallel applications competing for the throughput of a constrained link. The competing applications belong to the five application classes web browsing, file download, remote terminal work, video streaming, and Voice-over-IP. The results show that the proposed methodology improves the minimum and total utility, minimizes packet loss and queuing delay at bottlenecks, establishes fairness in terms of utility between applications, and achieves predictable application performance at high link utilization.

CCS Concepts: • **Networks** → *Network design principles*; • **Human-centered computing** → *Human computer interaction (HCI)*; • **Theory of computation** → *Linear programming*; • **Applied computing** → *Intranets.*

Additional Key Words and Phrases: SDN, QoE, QoS, pacing, enterprise, network, HAS, VoIP, browsing

Authors' addresses: Christian Sieber, c.sieber@tum.de, Chair of Communication Networks, Technical University of Munich, Germany; Susanna Schwarzmann, susanna.schwarzmann@inet.tu-berlin.de, FG INET, TU Berlin, Germany; Andreas Blenk, andreas.blenk@tum.de, Chair of Communication Networks, Technical University of Munich, Germany , Faculty of Computer Science, University of Vienna, Austria; Thomas Zinner, zinner@inet.tu-berlin.de, FG INET, TU Berlin, Germany; Wolfgang Kellerer, wolfgang.kellerer@tum.de, Chair of Communication Networks, Technical University of Munich, Germany.

ACM Reference Format:
Christian Sieber, Susanna Schwarzmann, Andreas Blenk, Thomas Zinner, and Wolfgang Kellerer. 2020. Scalable Application- and User-aware Resource Allocation in Enterprise Networks Using End-Host Pacing. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 1, 1, Article 1 (January 2020), 41 pages. https://doi.org/10.1145/3381996

1 INTRODUCTION

Increasing bandwidth demands by multimedia-rich applications and low delay requirements for real-time communications present a challenge for modern enterprise network designs. Despite a variety of demands, an enterprise network has to support the employees by providing a reliable infrastructure for the deployed network applications. Alongside the employees, the network resources are drained by automated processes such as backup transfers or by Internet of Things (IoT) devices such as surveillance cameras or sensors. A network design is required which allocates every application its share of the available network resources while at the same time minimizes the need for over-provisioning. There are three main challenging research questions for application- and user-aware resource allocation in enterprise networks:

I) *Define*: How to define an application-aware allocation of resources in terms of Quality of Experience (QoE) of the user, considering the variety of application classes and their demands?

II) *Determine:* How to determine shares of resources for each application under resource constraints considering the definition of application-awareness derived in I)?

III) *Allocate:* How to allocate each application its share of the network resources in heterogeneous enterprise networks where the availability of QoS mechanisms at each hop highly depends on the deployed switching hardware?

Today there are commonly two high-level approaches for resource allocation in enterprise networks: best effort transport with sender-based congestion control and Quality of Service (QoS) mechanisms on the forwarding devices. But this is neither stable or fair in terms of goodput when applications compete for a link's bandwidth [31], nor aware of the specific application or the user behind it. This can lead to bad application quality and, as a consequence, to user dissatisfaction. The second option, QoS configuration at the forwarding devices, either discards or delays data packets of an application or application class in favor of another class or application. However, enforcing QoS on intermediate devices has several drawbacks. Buffer space and scheduling QoS options are limited on the devices. Discarding packets along the way from the sender to receiver interacts badly with the sender's congestion control [14] and increases the network load due to the retransmission of discarded packets.

Moving the QoS enforcement from the intermediate devices to the end-hosts is a viable third option, as shown by data-center operators: By using a central controller, network monitoring, and programmable application pacing at the sender and receiver, a specific amount of the available throughput can be allocated to each application. Congestion in the network is then prevented by limiting the total sending rate of all applications [28]. At the end-hosts, applications, i.e., the primary contributors to the network load, can be restricted from generating more data than the network can carry. Furthermore, the limited QoS options, such as interface queues, can be reserved for high-profile use cases such as critical real-time traffic and separating managed from best-effort traffic.

In this paper, we apply this concept to enterprise networks and show that, indeed, a global control strategy with end-host pacing can significantly improve user experience. Next we define the problems in detail.

1.1 Problems Definitions

In plain best effort networks, resource allocation is implemented on transport-level at the endpoints, e.g., at web servers and browsers, via TCP congestion control. Congestion control works at sender-side by increasing or decreasing the sending rate based on observed packet loss and the Round-Trip Time (RTT). TCP's goal is to divide the available data-rate equally between active TCP connections. In the network, the data packets of a sending application, e.g., a web server, are treated equally by the forwarding devices. If the receiving rate at a forwarding device's interface exceeds the maximum physical sending rate, packets are queued in a buffer or dropped if the buffer is full.

The main problems with plain best effort networks are: (1) Some applications, such as web browsers, behave unfair and open multiple parallel TCP connections and therefore can receive a larger fraction of the available throughput. (2) Datagram-based applications, such as Voice-over-IP (VoIP), often do not implement any congestion control at all. (3) The effectiveness of TCP congestion control depends on factors such as the specific congestion control algorithm, delay, packet loss, relative start times of competing TCP flows and how active a TCP connection is. (4) Different demands of applications are not considered, e.g., in terms of minimum throughput and maximum delay. Thus, there is no application-awareness in best effort networks.

Commonly, the problems of best effort networks are addressed by enterprises by implementing QoS mechanisms in the network. QoS mechanisms on the forwarding devices allow to prioritize some packets over others based on matching rules. For example priority queuing allows to put VoIP packets based on the Type of Service (ToS) flag, VLAN tag or specific UDP ports into a queue with preferred treatment. That way the delay and packet loss of VoIP calls is kept low and isolated from other traffic. Flow- or class-based Weighted Fair Queueing (WFQ) allows to put individual application flows or whole application classes into separate queues with guaranteed minimum bandwidth. Token bucket (TB) policing allows to limit the data-rate of individual flows or classes without the need for switch buffer space. For example mobile service providers are known to use TB policing to limit the data-rate of video streaming services [14].

But implementing QoS in the network is costly and inefficient: (1) Buffers in forwarding devices are expensive and there is only a limited number of queues to configure per egress interface, typically about 8 [1]. This is insufficient for implementing a sophisticated strategy to distinguish hundreds of active applications of multiple classes in a network. (2) Policing interacts badly with transport-level congestion avoidance algorithms resulting in lost packets. Lost packets cause retransmissions and decrease transmission efficiency [14]. (3) Heterogeneous enterprise networks with diverse forwarding devices from different vendors are complex and error-prune to manage, hampering the enforcement of end-to-end QoS options. Furthermore, there are no common QoS abstractions across switching hardware vendors. Hence, deploying a single QoS strategy across devices might not be possible, especially if not all devices support the required features. (4) Encryption or header field ambiguity can prevent the correct identification of application classes in the network.

Hence, with limited or incompatible QoS mechanisms and the issues regarding identification of application flows, a scalable and application-aware network design is hard to implement in the network (see also Section 2).

1.2 Proposed Solutions

We realize the resource allocation by implementing centrally-controlled pacing of individual applications at the end-hosts. Packet pacing at the end-hosts ensures that a stream of packets conforms to a specified data-rate by adding artificial delays between consecutive packets during the sending process. Pacing prevents packet loss by smoothing out

[1] Jim Warner, https://people.ucsc.edu/~warner/buffer.html, last accessed: 11.10.2018

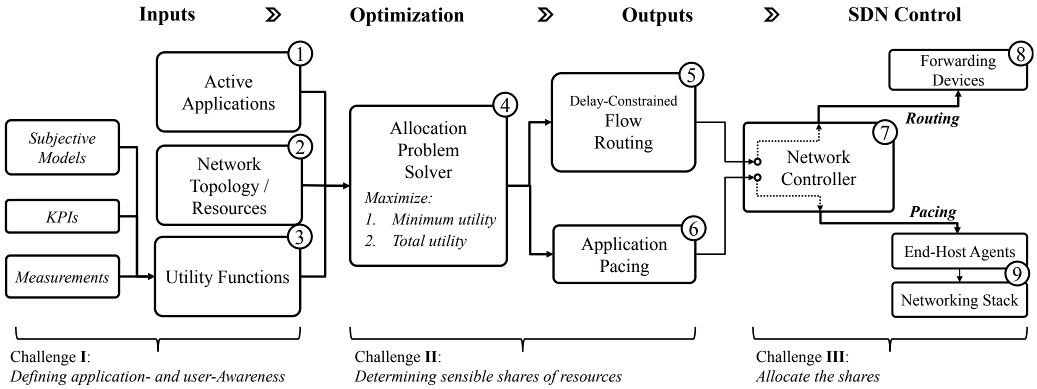


Fig. 1. Overview over the challenges and the proposed solution towards scalable application-aware resource allocation in enterprise networks. Based on the set of applications (①), resources (②) and application utility functions (③), the allocation problem solver maximizes the minimum and total utility over all active applications (④). As a result, delay-constrained flow routing (⑤) and application pacing rates (⑥) are implemented by a network controller (⑦) in the network (⑧) and on the end-hosts (⑨).

packet bursts and allows for shallow buffers in the intermediate forwarding nodes. Shallow buffers reduce queuing delays and avoid expensive switch buffer space. Applications can reliably determine their available goodput and it is unnecessary to probe the throughput by loss-based congestion control mechanisms. Furthermore, pacing at the end-host allows for implementation of effective backpressure to the applications producing the data, reducing the amount of buffered data in the network stack. Pacing at the end-hosts can scale to thousands of traffic classes [37], congestion in the network can be avoided by a central management of the available resources [28] and application flows can be identified at the source. Recent works show that bandwidth allocation to applications can be implemented hierarchically at global scale, enabling high percentages of link utilization [28]. Sender congestion control and QoS in the network are downgraded both to failsafe solutions and supportive roles in the overall QoS strategy, e.g., in cases the central control fails or embedded devices cannot be modified.

Ultimately, a user of an application does not care about what share of the resources is allocated to her/him as long as her/his user experience, or *Quality of Experience* (QoE), with the application is positive. For that reason, challenges I) and II), i.e., how to *define* and *determine* sensible allocations, are tackled based on the resulting user experience. We define the user experience as a per-application utility function of throughput and delay. The utility function is derived from user experience models from the literature and selected application Key Performance Indicators (KPIs). By jointly optimizing the utility and network resources usage, a fair share in terms of utility can be determined given a set of applications, utility functions, and constrained network resources. Challenge III) is the scalable allocation of the calculated application shares. We propose centrally-controlled application pacing at the end-hosts combined with per application flow routing. Routing is solved implicitly by our problem formulation by selecting paths for application flows which satisfy capacity and delay requirements. Routing per flow can then be implemented through Software-Defined Networking (SDN) for all applications in the network. The identification of application flows in the network, e.g., source and destination TCP/UDP ports, are provided to the central controller by software agents at the end-hosts. Applications can then be subjected to routing and pacing as dictated by the network controller.

Figure 1 summarizes the general methodology of the proposed solution. First, the active applications (①) in the network are determined by end-host agents and network monitoring. Second, the network topology and available resources (②) are known by the network controller. Third, a suitable utility function (③) based on subjective QoE models, application KPIs, and measurements is associated with each application. An allocation problem solver (④) then determines the per-application routing (⑤) and application pacing rates (⑥) based on a fairness criteria. Routing rules are then implemented by the network controller (⑦) on the forwarding devices (⑧) and pacing rates are enforced at the end-hosts (⑨).

The system is implemented as a proof-of-concept set-up with support for the following five application classes: web browsing, batch file transfer, VoIP, adaptive video streaming, and remote administration. In the set-up, we evaluate static scenarios where a fixed number of parallel clients with multiple applications have to use a resource constrained link to communicate with central services, such as it is the case in SD-WAN or remote building scenarios. The results show that central pacing can provide dependable application performance and increases inter-application fairness at high link utilizations.

1.3 Contributions

The contributions of this paper are as follows:

(1) We present a system design for scalable user-aware resource allocation in enterprise networks based on SDN-principles and end-host pacing (Section 3). The design does not make assumptions about the availability of QoS mechanisms such as WFQ on the forwarding devices.

(2) We define throughput- and delay-dependent utility functions for five application classes. Furthermore, we discuss deployment options and trade-offs regarding the creation and accuracy of the utility functions (Section 4). Compared to other works, the utility functions are based on actual subjective studies and thus tied to the experience of the user instead of technical KPIs. Furthermore, measurements of the applications' behavior under limited available resources are used to determine the relationship between resources and user experience.

(3) We formulate the utility throughput- and delay-aware allocation problem as a 2-step Mixed Integer Linear Program (MILP) with max-min fairness criteria. The first step maximizes the minimum utility in the network (*max-min-fairness*), while the second step maximizes the sum of all utilities for a constrained minimum utility (Section 5 and in detail in Appendix A). While the min-max utility proportional fair bandwidth allocation problem is well studied in literature, the problem combination of bandwidth allocation and delay-aware routing for arbitrary utility functions is not formulated so far. Note that in this paper we provide an optimal algorithm for the allocation problem, but with limited scalability.

(4) We evaluate application mixes with over 100 parallel applications of 5 common use cases in a proof-of-concept set-up. The results show how pacing can improve delay and packet loss at bottlenecks and can significantly increase inter-application fairness in terms of utility. Furthermore, pacing leads to predictable application performance even at high levels of network utilization (Section 7).

(5) We provide all material to the paper, such as the automated applications, a virtual experimentation set-up, and optimization formulation as open source software. [2]

1.4 Paper Structure

The paper is structured as follows. Section 2 introduces the background and related work. Section 3 presents the proposed system architecture. Afterwards we *define* the shares (Section 4), *determine* shares under resource constraints

[2] https://github.com/tum-lkn/appaware - Supplemental material to this article

(Section 5), discuss the *allocation* of the shares in a experimental set-up (Section 6) and *evaluate* the effectiveness of the proposed approach in the set-up (Section 7). Section 8 summarizes the results, discusses future research directions and concludes this paper.

2 BACKGROUND AND RELATED WORK

This section introduces fundamental network QoS control techniques and, from this, motivates the usage of pacing. Besides the technical basics, we describe its benefits and implementations, and present some works targeting pacing of individual TCP flows. Finally, we summarize related works on multi-application QoE management. We start by defining the term enterprise network in the context of this paper.

2.1 Enterprise Networks

Enterprise networks are not bound to the same net neutrality laws which govern most parts of the public Internet and access to an enterprise network can be limited to approved devices. The network operator is in full control of the applications deployed in the network and on the end-hosts. This is due to security concerns (e.g. malware, leakage of sensitive documents/data) and the need for performance guarantees for mission-critical applications. This means that end-hosts are restricted to a small set of applications, depending on the role of the employee, and that the communication of each application can be monitored. HTTP(S) traffic passes through a proxy to perform Deep Packet Inspection (DPI) to identify sensitive documents being uploaded on an external website or malware being accidentally downloaded.

The scale can range from small businesses housed in one building to global enterprises with multiple remote campuses connected to one or multiple central offices and millions of end-hosts. In order to adjust the available throughput according to the utility allocation, it is crucial to know or approximate the available throughput and to monitor the link utilization and packet loss. For delay-constrained routing per application flow and load balancing, Software-defined Networking with fine-grained flow control is necessary. If SDN control is not available, allocation can still be done based on the available forwarding graph, e.g., based on shortest-path routing. QoS control mechanisms on the network nodes are not required, but can be used to support the overall QoS strategy. For example, two VLANs in combination with two queues can be used to isolate managed from best effort traffic using hierarchical token bucket (HTB) scheduling.

2.2 Network QoS Control Mechanisms

On a basic level, QoS enforcement relies on two options of treating packets in the network: they either can be dropped or enqueued. Mechanisms that decide how packets are treated form the fundamentals of QoS control techniques, e.g., flow prioritization or rate allocation with weighted fair queuing are widely applied in today's communication networks [32]. Table 1 summarizes and classifies the most relevant techniques and gives state of the art examples. In the following, we shortly describe the listed mechanisms.

Active queue management (AQM) is applied within queues of network elements and describes the intelligent drop of network packets to control the queue length [35]. Excessively buffering packets causes bufferbloat and leads to increased delays. Random early detection (RED) [16] is one of the well-known and widely applied mechanisms for AQM. Conventional tail-drop mechanisms discard all incoming packets when the queue is full. RED drops incoming packets with a certain probability that increases with increasing queue length. To realize this, RED applies two thresholds: If the queue is (almost) empty, the probability to drop a packet is set to zero. If the queue is (almost) filled, all packets are definitely dropped. In between these two thresholds, the dropping probability increases linearly. That way, RED proactively prevents bufferbloat and reduces the bias of discarded packets against bursty traffic. Controlled Delay

(CoDel) [34] keeps the queuing delay of packets below a certain threshold. Packets are marked with the current timestamp as they enter the buffer. When dequeuing a packet, the CoDel algorithm computes the time it spent in the buffer. When the maximum delay, by default 5 ms, is exceeded for a certain amount of time, subsequent packets are dropped at the head of the queue. In contrast to CoDel and RED, Explicit Congestion Notification (ECN) [15] does not proactively discard packets. Instead, it marks packets in case of impending congestion to inform the receiver, which in turn signals the impending congestion to the sender. As the ECN-aware endpoints adapt the sending rate accordingly, ECN performs the queue length control and bufferbloat prevention in an indirect manner.

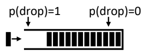
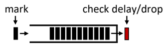
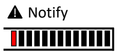
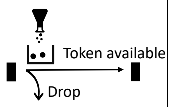
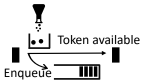
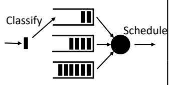
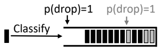
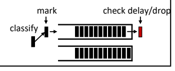
Rate limiting mechanisms manage queues or flows to achieve a target traffic rate. One rate limiting example is policing, which controls the rate of a flow by dropping network packets. This is realized by applying token bucket or leaky bucket algorithms. Tokens are created according to the target rate. If not enough tokens are available, packets to be sent are dropped. In contrast to policing, where packets are dropped in case that no tokens are available, shaping enqueues packets and allows them to wait for a token to be created.

Scheduling algorithms decide about how packets are dequeued when several queues are active. Hence, they operate between queues. First of all, incoming packets are classified based on pre-defined QoS policies and are accordingly inserted into one of the queues. The scheduling algorithm then decides about the order and frequency with which packets can be released from the different queues. Allowing certain queues to transmit packets more often than others, enables QoS enforcements in the sense of allocating higher bandwidth shares, i.e., different priorities to different queues. Such mechanisms that govern how packets are queued and de-queued, are often referred to as *queueing disciplines (qdiscs)*. They can further be categorized as handling packets in either a classful or a classless manner. We omit the differentiation in Table 1, but shortly emphasize the difference in the following. Classless queueing disciplines are well suited for basic traffic management and come with decreased configuration overhead compared to classful queueing disciplines. Classful qdiscs allow a more differentiated treatment of different kinds of traffic on the costs of increased configuration efforts, like the definition of appropriate filters and classes. From the examples above, Class-based Queueing (CBQ) [17], Hierarchical Token Bucket (HTB), and Weighted Round Robin (WRR) fall within the classful qdiscs, while Round Robin (RR) is an example for a classless qdisc.

The paradigm of smart queue management combines active queue management and scheduling. Weighted Random Early Detection (WRED) [44] allows to apply several thresholds of dropping packets in one queue. For example, while packets of one QoS class are dropped if the buffer is half filled, the packets belonging to another QoS class are only dropped if the buffer is completely filled. Furthermore, WRED supports applying several queues with different buffer lengths. On the one hand, this allows to additionally influence the packet dropping probability for different QoS classes. On the other hand, scheduling between the queues enables realizing further QoS policies, like packet prioritization. Flow Queue Codel (*fq_codel*, RFC8290) is an extension of CoDel. It uses multiple queues, whereby each of the queues employs CoDel. A scheduler decides based on a modified Deficit Round Robin algorithm, from which queue a packet should be dequeued. *fq_codel* allows to enforce QoS policies by classifying the packets and allocating them accordingly to queues.

Although many QoS enforcement mechanisms exist and are applied in today's networks, they cannot be straightforwardly applied in our case. Some of the techniques listed in the table are not powerful enough. ECN, for example, is capable to influence the sender's rate to prevent packet loss, but does not allow to set a specific rate. The major drawback when it comes to applying those mechanisms is the limited number of configurable queues in network

Table 1. Overview of traffic QoS control/allocation techniques applied in communication networks

Location	Technology	Action		Example	Description	Illustration
		Drop	Queue			
Within queues	**Active queue management:** Manage the queue length	X		RED	Drops packets based on statistical probabilities instead of conventional tail drop. Prevents high delays resulting from full buffers.	
				CoDel	Reduction of packet transmission delays by preventing large and constantly full buffers	
				ECN	Notification about network congestion without dropping packets	
	Rate limiting: Achieve target traffic rate	X		Policing	Tokens are created with a rate corresponding the target traffic rate. If no tokens are available, incoming packets are dropped.	
			X	Shaping	Tokens are created with a rate corresponding the target traffic rate. If no tokens are available, incoming packets are enqueued.	
Between queues	**Scheduling:** Allocate resource to queues		X	RR	Round Robin lets every active data flow take turn in transferring packets on a shared channel in a periodically repeated order.	
			X	CBQ	Divides user traffic into a hierarchy of classes and performs class based queueing so to allocate bandwidth to traffic classes.	
			X	WRR	Allows to differentiate QoS classes by allowing certain queues to put more packets on the wire.	
			X	HTB	Hierarchical token bucket allows for setting bandwidth thresholds to different flow classes.	
Hybrid: Within and between queues	**Smart queue management:** QoS-aware queue mangagement	X	X	WRED	Supports several queues that vary in buffer size and allows several thresholds per queue. Packets of higher prioritized flows are less likely to be dropped.	
		X	X	FQ-CoDel	Flow queue CoDel (fq_codel) extends CoDel by applying several queues. Allows for differentiating QoS classes.	

elements such as switches and routers.[3] As a consequence, QoS can only be enforced on aggregated flows and QoS classes. Hence, the limited scalability hinders a fine-granular QoS control. Shifting the QoS enforcement from network nodes to the end hosts constitutes a scalable method that allows for fine-grained QoS control. For that reason, we propose to apply TCP pacing to enforce traffic rates on a per application basis.

2.3 Pacing

The term pacing is used in different contexts and it is important to distinguish where it is applied and who is dictating the pacing rate. There can be pacing per interface, per application and per network socket or a combination of all three. The pacing rate can be set autonomously, e.g., like in TCP pacing, or by an external entity, e.g., by a central network controller. The term TCP pacing is an example where the rate is set autonomously and refers to the technique where the packets of one TCP transmission window are spread out over the measured RTT [2]. The target pacing rate is

[3] Jim Warner, https://people.ucsc.edu/~warner/buffer.html, last accessed: 11.10.2018

determined by the congestion control algorithm based on observed packet loss or delay. In this work, when we use the term pacing, we apply pacing per-application flow and it is set by the central network controller. One application flow can include one or multiple streams (TCP) or datagram-based (UDP) transmissions which share the same source, destination and network path. Hence, all packets sent by the sockets of an application flow have to share the allocated pacing rate. In the following, we shortly introduce the pacing implementation of the Linux Kernel. Afterwards, we highlight the advantages of this technique compared to other rate limiting approaches, i.e., policing and shaping. Finally, other works relying on pacing are summarized.

2.3.1 Pacing Implementations. Pacing follows the approach of placing gaps between outgoing packets so to evenly space data transmissions [2, 8]. In the case of the Linux pacing implementation, the departure time of the next packet $time\_next\_packet$ is determined by the current time *now*, the size of the current packet *pkt_len*, and the target pacing rate $target\_rate$:

$$time\_next\_packet = now + \frac{pkt\_len}{target\_rate}$$

For details on the technical fundamentals and the way pacing is applied in this work, please refer to Section 6.2. Google is currently putting much efforts in developing efficient, rate-compliant, and scalable traffic control mechanisms, mainly for a deployment in data centers. To do so, they implement pacing in many of their recent approaches. With *TIMELY*, they propose an RTT-based congestion control [33]. Their congestion-based congestion control (BBR) [6] is implemented in the Linux kernel and used by all Google and YouTube server connections. With *Carousel* [37] they present a scalable traffic shaping mechanism by controlling packet release times where the target rate can be set by external entities per traffic class. As we do not have those strict requirements on scalability as for *Carousel*, we apply a custom version of the Linux *fq* implementation (Section 6.2).

2.3.2 Benefits and drawbacks of pacing. Pacing eliminates several drawbacks of other strategies for traffic rate control. While policing drops packets exceeding the target rate and shaping enqueues those packets, pacing follows the approach of delaying packets so to reach a certain rate. On the one hand, this eliminates the problem of increased overall network load resulting from retransmitting dropped packets of policed flows. On the other hand, there is no RTT inflation, as with shaping. Policing interacts poorly with TCP, as a result, policed flows suffer from low throughput even at low packet loss rates [14]. In contrast, pacing can increase the link utilization in shared environments. Delaying the outgoing packets at the sender in a controlled manner reduces burstiness, which implicates less packet loss and results in fewer triggers of TCP's congestion control. Furthermore, configuring target rates at end hosts brings the advantage of scalability, compared to other techniques. By shifting the QoS control to the involved end-hosts, pacing facilitates a fine-grained control on flow- and application-level.

However, studies have also shown that this only applies to some cases, while even with all-paced TCP flows the performance is worsened in many cases [2, 20]. According to the authors of the studies, this can be attributed to the fact that TCP pacing delays the congestion signal and that pacing results in synchronized packet drops. A pacing system that shapes traffic under consideration of the buffer queue is proposed in [5]. The authors introduce Queue Length Based Pacing (QLBP) to shape the traffic at access networks, so to smooth the traffic before entering the core network. This is especially interesting for small buffer networks, where packet loss is more likely to occur. The reduced packet loss, as a consequence of the decreased burstiness, results in a nearly fully link utilization when using the proposed solution. The QLBP algorithm is also applied in [4] to study the impact of pacing on different network traffic conditions. The authors conclude that pacing is especially beneficial in networks with small buffers, where packet loss, as a result

Table 2. Summary of state-of-the art approaches that make use of TCP pacing, along with their scope and the entity deciding about the pacing rate.

Technique	**Category**	**Scope**	**Rate set by**
BBR [6], TIMELY [33]	TCP Pacing	One TCP socket	TCP congestion control
Carousel [37]	Efficient and scalable pacing implementation	Flexible, based on traffic classes, evaluated per flow	External controller or TCP congestion control
FQ[4]	Linux kernel pacing implementation	Per flow	Primarily congestion control algorithm, can be manually overwritten
BwE [28]	Hierarchical bandwidth allocation	From global to per computing task	External controller
QLBP [4, 5]	Edge pacing	Single queue per interface	Adaptive based on queue-length
Our work	End-user application pacing	Per a subset of application's sockets	External controller

from bursty traffic, can significantly reduce network performance. They furthermore show that pacing can have a small negative impact on short-lived flows, if the parameters are not set appropriately. Finally, it is shown that the fairness achieved by pacing only slightly differs from the fairness as achieved by TCP. The performance of host traffic pacing and edge traffic pacing, i.e., pacing the traffic before it enters the core network, is compared for small buffer networks in [19]. The results indicate for most of the evaluated scenarios that edge pacing performs at least as well as host pacing in terms of link utilization. Edge pacing also has practical benefits, as it does not require an adaptation of the involved clients. A critical analysis on pacing is performed in [43]. The authors evaluate the impacts of pacing for several TCP implementation and scenarios. They conclude that the benefits when applying pacing depend on the used TCP implementation and on the performance metrics that are relevant for a specific application. However, due to the tendency to high speed protocols, they predict an increasing motivation to use pacing in future. Furthermore, they showed that in some cases, pacing was capable to improve the performance of both, paced and un-paced flows. As a drawback, the work highlights the unfairness among paced and non-paced flows in terms of bandwidth, as paced flows do not receive their fair share when competing with non-paced flows.

We summarize the approaches that rely on TCP pacing in Table 2. It shows that pacing is applied on per-interface, per-flow, and per-task levels, but so far not on a per-application level. Although these approaches might provide fairness on a per-flow level, they do not provides fairness between applications (10 connections opened by a web browser vs. 1 connection for a file download) and it still remains unclear how this could be applied to UDP-based applications where there is no operating system support for TCP-style probing of the available throughput. This work aims at closing the gap of considering pacing from an application-centric perspective, i.e., to evaluate its feasibility for application-aware network management. We investigate the conformance of actual rates and delays to the target values, which dictates the degree of granularity to which QoE can be controlled. As we will find that pacing constitutes a feasible method to do so, we present a proof-of-concept architecture for optimizing QoE fairness in a multi-application environment.

Table 3. Overview of related works targeting multi-application QoE-awareness and their classification in terms of utility function, determination of QoE-aware resource shares, and allocating of determined resources. 0 denotes that no utility functions are applied at all, + denotes that utility functions are applied mapping either AQoS or AQoS plus NQoS to QoE, ++ represents utility functions solely relying on configurable network resources, e.g. bandwidth or PSNR in radio access networks.

Source	**Utility function**		**Determine**	**Allocate**
	Classification	Description		
[21]	+	Mapping AQoS to QoE	Not specified	Generic control concepts
[40]	++	Mapping NQoS to QoE	Particle swarm optimization (PSO) based algorithm	Applying the proposed algorithm to resource block allocation technique in LTE
[36]	++	Mapping NQoS to QoE	Game theoretic approach	Radio resource management applying proposed game theoretic approach
[30]	+	Mapping NQoS and AQoS to QoE	Optimization based on multi-choice knapsack problem (MCKP)	Carrier scheduling applying proposed optimization algorithm
[11]	++	Mapping of network bandwidth to QoE	Solving multi-objective optimization problem	Joint subcarrier and power allocation scheme
[12, 13]	0	Application feedback instead of utility functions	Not specified	Admission Control, bandwidth guarantees
[38]	0	Hypothetical utility functions mapping NQoS to QoE	Proposed algorithm optimizing bandwidth allocation	WFQ scheduling with QoE-optimized weights
[18]	++	Mapping screen resolution and bitrate to SSIM	Branch and bound algorithm to find optimal set of video bitrates	Video bitrate guidance for heterogeneous clients
[28]	+	Mapping bandwidth to an arbitrary fair share	Novel Multi-Path Fair Allocation (MFAA) algorithm	Enforced via pacing at the hosts

2.4 Related Work on Multi-Application QoE Management

Several efforts have been made towards QoE-awareness in multi-application scenarios. Some relevant approaches are summarized in Table 3. The first column denotes the investigated approaches. The remaining table columns represent the three challenges introduced beforehand: *define*, *determine*, and *allocate*. However, we found that none of the reviewed work explicitly defines the required resources to obtain a certain Mean Opinion Score (MOS) [4], but they all utilize in some form utility functions that map application QoS (AQoS) and/or network QoS (NQoS) to express QoE. The MOS scale describes the experience of a user with the application on a scale of one to five where the scale is labeled with {Bad, Poor, Fair, Good, Excellent}. For that reason, we replaced in the table the *define*-step by a classification and a short description of the applied utility function. In the following, when reporting on related work, we focus on how and which utility functions have been applied, how the appropriate resource shares are determined, and the applied methods to allocate the resources.

BwE[28] introduces a global hierarchical top-down bandwidth allocation scheme used in Google's internal network for distributed computing tasks. Bandwidth allocation is done via a function that maps bandwidth to a "relative priority on an arbitrary, dimensionless measure of available fair share capacity". The BwE reference is important as it shows that global and large-scale bandwidth allocation is indeed possible in production environments. But how to derive an allocation for end-user applications and how they benefit from it, is not discussed in BwE. In contrast, this paper at hand focuses on end-user applications and the interplay with, and possibilities for, network control to guarantee a specific user experience to the end users.

[4] https://lwn.net/Articles/564825/

[4] ITU-T Recommendation P.800. Methods for objective and subjective assessment of quality

Many related works with a focus on multi-application QoE-aware networking are associated to the mobile domain [11, 21, 30, 36, 40]. Several KPIs are proposed to be monitored at network elements in the architecture of [21], including packet loss rate, throughput, and RTT. At the clients, network-related parameters, e.g., delay, and application-based metrics including web page download time or video buffer and bit-rate can be measured. The collected AQoS metrics are used to estimate the per-application QoE using models from literature. One of the presented use-cases in [21] considers a QoE optimization based on the estimated QoE values. To do so, the authors list a variety of parameters that can be configured along the protocol stack in order to control QoE. The work does not provide a specific algorithm for determining the required resources, nor does it propose a designated method for allocating them. Instead, the authors outline several possible control actions like bandwidth limiting or QoE-aware capacity planning. As the applied utility functions only rely on AQoS, the QoE can only be controlled in a qualitative manner, meaning enhancing and degrading the QoE, but not controlling it so to achieve a specific MOS value.

Tang, et al. [40] proposes a novel algorithm for resource block allocation in LTE systems to maximize QoE whilst preserving fairness among users. The authors also use existing models to estimate user QoE, but adapt the models so to express the mean opinion score (MOS) solely from network-parameters like delay or packet error probability. Based on these models, the authors present a resource block allocation algorithm that is based on Particle Swarm Optimization.

Another QoE-aware resource scheduling algorithm for mobile networks is based on a game-theoretic approach [36]. The QoE is estimated for various applications using models from literature that map network parameters to MOS. The users' data flows cooperate with each other in a proactive manner and jointly optimize the QoE in a game-theoretic based manner. Instead of using the conventional throughput maximizing algorithm in radio resource management of OFDMA, the authors propose to implement their scheduling algorithm which aims on maximizing the fairness among heterogeneous users.

The approach described in [30] targets QoE-awareness in mobile LTE-Advanced networks. In the QoE modeling step, both NQoS and AQoS are used for estimating the user perceived quality for different application types. The estimated QoE and available bandwidth are inputs to the resource scheduling algorithm, which solves a multi-choice knapsack problem (MCKP) that maximizes the sum of all users' MOS values. The component carriers are dynamically scheduled according to the network traffic load by this QoE-aware scheme.

A further approach towards QoE-driven resource allocation in wireless networks is [11]. The authors apply utility functions which express MOS for various applications as functions of different NQoS parameters. Thereby, they assume a packet error probability of 0, a packet loss rate of 0, and fixed frame rate in the case of video streaming applications. Using these simplified utility functions, the authors propose a solution to a multi-objective optimization problem which aims at maximizing MOS. As network resource control mechanisms, the authors apply an efficient allocation of subcarriers among the active users.

The concept of Participatory Networking is proposed in [12, 13]. It describes an API that can be used by applications, end-hosts, and devices to interact with the network. A centralized controller is authorized to delegate read and write access to the network participants. Using the write access, applications, users or end-hosts can reconfigure the network according to their needs and can provide knowledge to the network, e.g., their future traffic demands. Hence, no utility functions that map AQoS or NQoS to QoE are needed, as the application instances directly communicate their requirements to this controller.

[38] applies hypothetical piecewise linear functions that map bandwidth to QoE and propose a new scheduler for fair and efficient bandwidth allocation in shared networks. Using these utility functions, they optimize the bandwidth per flow so to have a fair utility over all active applications. According to the bandwidth shares, the weighted fair scheduler

allocates respective weights to the flows. Simulation results show that the minimum utility can be increased significantly, while maintaining the same average utility in most of the cases, compared to a conventional max-min-fairness approach.

[18] presents an SDN-based framework to support a fair video QoE for all clients within a shared network. The utility function maps a client's device resolution and bitrate to structural similarity (SSIM) [42]. Considering the current network capacity, a controller decides about the bitrate for each video client, so to provide a similar quality to each of them. The bitrates are communicated to the streaming clients, which in turn request the respective quality level from the video content server.

The presented strategies are all steps towards QoE-awareness in multi-application systems. Some of the works rely on state of the art control mechanisms, but propose novel resource scheduling or allocation techniques. However, the applied utility functions often depend on features, which cannot be influenced in a direct manner. As a result, those approaches allow for a qualitative, less targeted QoE control. For example, a low video quality implies a low MOS value. Providing more bandwidth will enhance the playback quality and increase MOS, but it is not possible to quantify the impact of providing a certain amount of bandwidth on MOS scale.

We present an approach that allows to quantitatively map the NQoS parameters bandwidth and delay to MOS. Furthermore, we propose to apply network pacing, which allows us to control both, the bandwidth allocated to a flow and the end-to-end delay. Having utility functions which only rely on controllable parameters allows for a targeted, fine-grained QoE optimization.

3 SYSTEM DESIGN

The background on multi-application QoE architecture designs shows that previous proposals cannot combine accurate identification of application- and user-aware resource demands with scalable resource allocation. In the following we propose a new design considering the following aspects: a) Awareness of the active applications in the network and their demands, b) per-application resource allocation and c) per-application forwarding for delay- and capacity-constrained routing. Our design relies on a centralized control, whose major drawback is the introduction of a single point of failure (SPOF). Although not addressed in detail in our work, we would like to emphasize that several approaches exist for physically distributed, but logically centralized SDN control planes, to overcome the SPOF problem. Proposed solutions either rely on hierarchical [22] or flat organizations [41] and enable scaling the load among several controllers.

Figure 2 illustrates the system design. A logically centralized network controller (①) exerts control over the forwarding devices, the *data-plane*, by applying network control (②) through SDN protocols. SDN protocols, such as OpenFlow, enable per-flow routing by pushing simple match-action rules to SDN-enabled devices. Resources are allocated in the network by pacing ($\boxed{\text{P}}$) the data-rate sent by traffic sources into the network (⑤). Pacing is applied at the edges of the network, i.e., at end-hosts such as clients and servers, or at gateways (⑦). Software agents on the end-hosts (④) allow the network controller on the one side, to know about which applications access the network, and on the other side, to apply pacing to the applications (③) at the host's networking stack $\boxed{\text{P}}$. Applying pacing at the end-hosts' networking stacks can support tens of thousands of individual flows with low additional resource consumption for the host [37]. All conversations in the network are subject to the pacing set by the network controller. If a conversation cannot be paced at its networking stack, pacing can then be applied, for example, at the first hop in the network. Delay requirements are fulfilled by selection of appropriate links and target link utilizations. We describe how pacing is implemented in our set-up as part of the experiment design in Section 6.2.

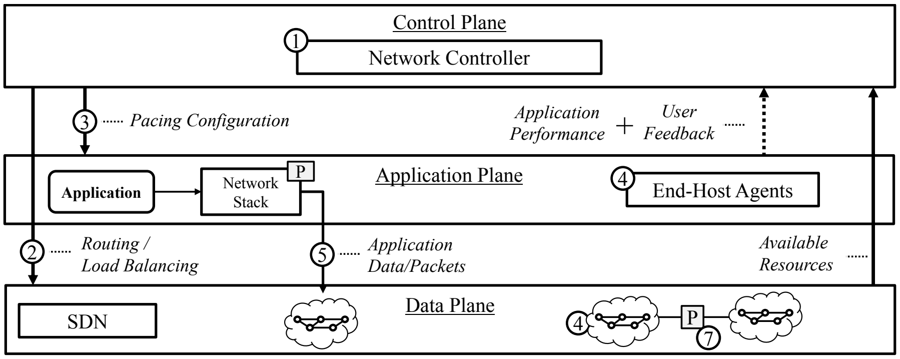


Fig. 2. Overall system design. A logically centralized network controller (①) provides per-application delay-constrained routing and resource allocation. Applications are identified by software agents at the end-hosts (④) and resources are allocated through restricting the total sending rate of applications at the hosts' networking stacks (③, $\boxed{P}$) and at the network edges (⑦). Per-application routing is implemented by using SDN protocols to push individual forwarding rules to the network devices (②). Per-application delay requirements are fulfilled by a careful selection of the flow path and target link utilizations.

3.1 Intents and Intent Hierarchy

Some application flows transported on the network, such as system monitoring or building surveillance, have predictable traffic patterns and determining a suitable data-rate to allocate is straightforward. Furthermore, periodic background jobs, like backup data transfers, can be scheduled based on the approximate amount of data and the deadline for completion. However, for user-facing applications, the variety of demands is higher. Determining the appropriate pacing data-rate for such applications is challenging. It is insufficient to consider only the *class* of an application, e.g., web browser, but also for what purpose the application is used. For example, modern web browsers are an execution environment for a variety of business applications, from employee and financial management to video streaming (*DASH*, *HTMLMediaElement*) and video conferencing (*WebRTC*). Hence, we distinguish between application *classes* and application *intents*. An intent can be specific, such as a video stream of a surveillance camera with specific encoder settings, or broad, such as general web browsing. A running application can also participate in multiple conversations with different intents and conversation endpoints. Hence the resource demands of a conversation are defined by the tuple of *(class, intent)*.

Identifying the intent of an application accurately enables the specification of precise application demands and the selection of suitable user experience models. Both are essential to implement predictable application performance and to improve accuracy in terms of QoE for the user. We argue that in an enterprise deployment, a holistic identification of all classes and intents is infeasible. Therefore, we propose a hierarchy of intents as illustrated by Figure 3. The figure shows a possible enterprise intent hierarchy by example. At the root of the hierarchy, there is a default intent which offers basic guarantees in terms of throughput and delay to unidentified applications. The root intent is followed by the application classes such as video streaming and remote terminal work.

Intents can be specified with an arbitrary hierarchy depth. If an application's intent cannot be identified accurately, a higher-level intent can be selected. However, this comes with the cost that the allocated resources do not fit to the targeted application performance. For example, the hierarchy in the figure specifies the two common voice codecs G.729 and G.711 as sub-intent for desktop VoIP-phones. If the codec is known, e.g., based on the MAC/IP address of the phone or from a database, the demand and user experience model are well-defined. If the conversation from the phone can

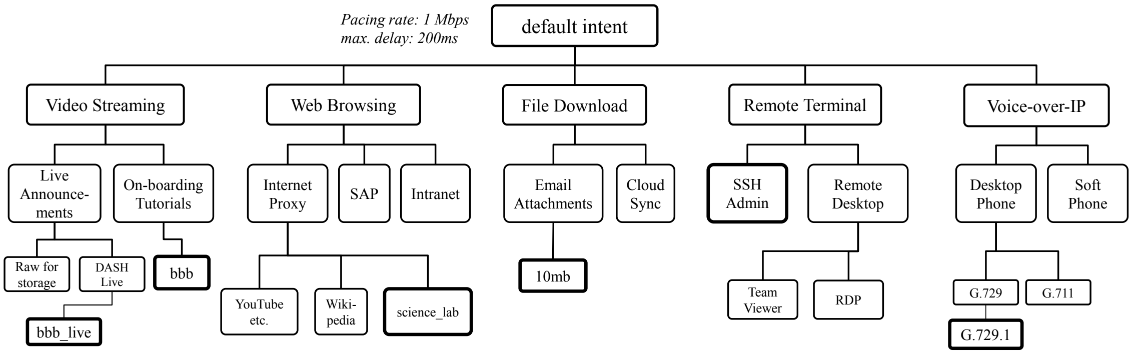


Fig. 3. Illustration of a possible hierarchy of application classes and intents. Classes and intents specify the utility function and user experience model to use for the target utility calculation and resource allocation. There is a trade-off between predictable application QoE and the effort for the company to construct the hierarchy. If an application's intent cannot be identified, fall-back rules can be applied to select a higher-level intent with the cost of reduced QoE accuracy. Highlighted intents are part of the evaluation.

only be identified as a desktop phone, one may define the highest known demand from all codecs. How to create such a deep hierarchy is out of scope of this paper. We restrict our hierarchy to the five classes and six intents as highlighted in bold in the figure. One can imagine that a combination of user-feedback and network/application monitoring, combined via machine-learning and some manual work, results in an accurate representation of the enterprise environment.

3.2 Network Controller and Application-Awareness

The question remains how applications can convey their class and intents to the network controller. We propose local agents (⑤) at the end hosts as an interface between applications and network control. There are two basic options from here. The first one is to modify the applications to report their identity and intent(s) to the agent. This can be done with a standardized API, for example through client or server extensions. The agent then forwards this information to the network controller and waits for the controller to decide on the appropriate pacing rate to apply. The second option could be for the agent to monitor connection establishment or perform Deep Packet Inspect (DPI) and classify the conversations by matching it to known endpoints, header fields, packet payloads, process names or function calls. Other techniques from the area of application performance management (APM), like code injection or tracing in the operating system, could also be used.

Without installing an agent, you could also capture packets of a conversation at the first hop, for example by using the SDN protocol OpenFlow and its *packet_in* feature [5], or by custom middle-boxes. However, in the network most traffic is encrypted and identification becomes difficult. There is no one-size-fits-all solution for how to identify applications and their intents as the available options depend on the specific enterprise environment. One can expect that a combination of rule- and pattern-based matching and machine-learning reduces the required manual work to a minimum.

3.3 Utility Functions

Once the application class and intent of a conversation are identified, the controller looks up the utility function for the *(class, intent)* tuple from a database. The utility function describes the relationship between demand, in terms of minimum throughput and maximum delay, and benefit, in terms of utility. We define utility as a dimensionless unit in

[5] http://flowgrammable.org/sdn/openflow/message-layer/packetin/, last accessed: 11.10.2018

Table 4. Applications, Intents and Key Performance Indicators

Class	Application	Intent(s)	Shorthand(s)	KPI(s)	QoE Model
Web Browsing	Firefox, selenium[7]	*science_lab*	*WEB*	Page Load Time	Egger et al. [10]
File Download	Python *requests*	*emailattach*	*DL*	Download Time	Egger et al. [10]
Video Streaming	TAPAS [9]	*bbb*, *bbb_live*	*VoD*, *Live*	Average Quality	*custom*
Remote Terminal	SSHv2, paramiko[6]	*sshadmin*	*SSH*	Response Time	Casas et al. [7]
Voice-over-IP	D-ITG [3]	*g729.1*	*VoIP*	Delay, Loss, (+ Jitter)	Sun et al. [39]

the range of [1, 5], which describes the satisfaction of the user with the service. The subsequent Section 4 introduces the utility functions in detail.

4 UTILITY FUNCTION DEFINITION

Comparing the performance of different applications with conceptual different KPIs requires mapping functions to a common scale. We denote the scale as *user-aware utility scale* and we define it with a dimensionless quantity in the range of [1, 5]. The utility functions then describe the relationship between the amount of resources allocated to an application and the resulting experience of the user with the application. In the following section we define the utility functions for selected classes of applications and intents. First, we present the considered application classes, intents, and KPIs of the deployed implementations. Second, we discuss the selected user experience models from the literature. Third, we define the utility functions based on measurements and the user experience models.

We consider five application classes: Web browsing, file download, video streaming, remote terminal work, and Voice-over-IP (VoIP) (Table 4). *Web browsing* covers a wide range of use cases, as modern web standards facilitate the move from proprietary and platform-dependent software to responsive web applications running in the browser. *File download* is the batch-transfer of data the user is waiting for, such as an email attachment. Use cases for adaptive *video streaming* in the enterprise range from announcements to training videos, such as on-boarding lectures for new employees. Depending on the purpose, both, video-on-demand as well as live transmissions, are conceivable. In particular major announcements are taxing for the infrastructure when viewed by a large fraction of the staff in a short time-frame. *Remote terminal work* by secure shell access allows administrators to access the terminals of servers, hosts, and switches from anywhere. The application class *VoIP* includes office phones, conferencing by software or in the browser, and VoIP applications on smartphones. We denote the combination between an application class and intent as application *type* and use the types *WEB*, *DL*, *VoIP*, *Live*, *SSH* and *VoIP* as shorthands for the investigated combinations of application classes and intents.

4.1 Applications, Intents and KPIs

Next we discuss the implementations, KPIs, and intents per application class in detail. KPIs in parentheses in Table 4 are not inputs for the user experience models, but are part of the evaluation in this paper.

4.1.1 Remote Terminal Work. For remote terminal work we define the intent of an administrator typing commands over a Secure Shell (SSH) connection. An automated SSH client enters commands and measures the duration until the output of the command appears in the terminal. Only commands which require minimal processing on the server-side, e.g., *uptime* and *date*, are entered. The SSH connection is established before the start of the experiment. OpenSSH 7.2 is used as server implementation on Ubuntu 16.04.4 LTS systems. Client-side automation is implemented using *paramiko*[6].

[6] http://www.paramiko.org/, last accessed: 11.10.2018

4.1.2 File Downloads / Web Browsing. File download is the batch transfer of a chunk of data over one TCP connection. As intent we define *emailattach*, a file with random content and a size of 10 MB, which is placed on an HTTP server for download. In an enterprise environment this intent could represent the maximum size of email attachments. The download is implemented using a short Python script and the *requests* library. As KPI, the script measures the duration from when the GET request is sent, up to the last received Byte.

Web browsing is implemented using Firefox in version 58.0.2 automated with *selenium*[7]. The settings are left to the default state and the cache is cleared after every page view. The number of parallel connections is limited to six per server and HTTP pipelining is not supported anymore by recent Firefox versions. The connections are configured to be persistent between requests. The browser interface is disabled (headless mode) and no page rendering is performed in the experiments to minimize the influence of system load and deployed testbed hardware.

This is a scenario where a limited number of browser-based business applications are used frequently and/or all web browsing sessions are tunneled through an enterprise proxy. With proxies, connections can be persistent even when requesting content from different domains. General web browsing, where multiple domains are involved without proxy, is not represented well by assuming persistent connections. This is due to the fact that connection establishment can significantly influence the page load time for longer transport delays. We define the KPI for one web browsing request as the duration from the initial GET request to the time all embedded resources are received (*page load time*). For web browsing we define the intent *science_lab*. The science_lab [2] template is a web-site with 22 objects with a total size of about 1.3 MB.

4.1.3 Adaptive Video Streaming. HTTP adaptive video streaming is implemented using the TAPAS[9] DASH player. The *conventional* [29] bit-rate adaptation strategy is selected. We consider one video view as one request and select the average quality level of all downloaded segments as KPI. We define the intent *bbb* for on-demand video streaming. For this intent, we encode the open-source movie Big Buck Bunny in six quality levels with average bit-rates of 486 Kbps, 944 Kbps, 1389 Kbps, 1847 Kbps, 2291 Kbps, and 2750 Kbps. Only the first 60 s of the movie are selected and segmented into 15 chunks of 4 s each. The playback buffer is configured with a maximum size of 60 s

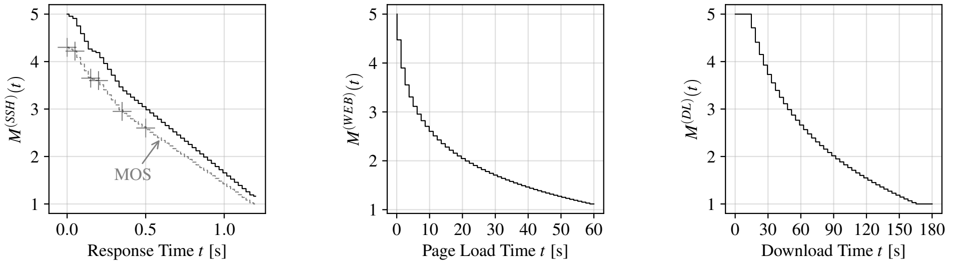
Additionally, we define the live-streaming intent *bbb_live* where the chunk size is reduced to 1 s and the buffer is limited to 10 s. Due to encoding overhead for the shorter chunk duration, the bit-rates increase to 572 Kbps, 1103 Kbps, 1625 Kbps, 2145 Kbps, 2660 Kbps, and 3172 Kbps.

4.1.4 Voice-over-IP. We emulate VoIP traffic using the Distributed Internet Traffic Generator (D-ITG) by Botta et al. [3]. D-ITG reproduces the inter departure-times and packet sizes of VoIP traffic and measures the KPIs jitter, packet loss, and delay of the resulting UDP packet stream. We define the intent *G.729.1* for VoIP and configure D-ITG to emulate RTP VoIP calls with the codec G.729.1. In this configuration, a constant bit-rate stream with 50 packets per second is generated with a packet size of about 20 Bytes ($\approx$ 8 Kbps).

4.2 Utility from KPIs

We define the current utility value of an application as an estimation of the instantaneous satisfaction of a user with the interaction with the application. The relationship between KPIs and user experience has to be determined through subjective studies, either directly by conducting dedicated laboratory, field, or crowd-sourcing studies, or indirectly by measuring user-relevant success metrics such as task completion times. We denote this relationship as M: KPI $\mapsto$ Utility.

[7] https://www.seleniumhq.org/, last accessed: 11.10.2018



(a) Terminal Work ($M^{(\text{SSH})}$) (b) Web Browsing ($M^{(\text{WEB})}$) (c) File Download (10M) ($M^{(\text{DL})}$)

Fig. 4. Utility from application KPIs (M: KPI $\mapsto$ Utility) derived from subjective study results scaled to range [1, 5]. $M^{(\text{SSH})}$ is derived from subjective study [7, Fig. 5 (a)] by Casas et al. Plus signs indicate the MOS data points as collected by the authors in the study. Web and file download utility values are derived from subjective user studies in [10] by Egger et al.

In case there is a suitable QoE Mean Opinion Score (MOS) model available for the application based on subjective studies, we take a scaled version of the MOS model for M. Thus, the utility functions are based on the average user experience of the test subjects in the referenced studies. However, the range of some user experience models does not reach up to 5.0 (Excellent). In those cases, we define the M by scaling up the experience model to $[1, 5]$. If no model is available, we define M based on hand-picked application KPIs.

QoE is an active area of research and holistic models do not exist yet for most applications. There could be alternatives or more complex models available for the selected user experience models. Furthermore, custom enterprise applications might require custom user experience studies. In any case, the presented system design and findings of this paper are independent of the concrete deployed user experience models. Therefore, the selected models in this work should be seen as rough approximations of the true underlying user experience.

4.2.1 Remote Terminal Work. We piece-wise interpolate M for remote typing from the results presented in [7, Fig. 5(a)]. There, Casas et al. study the QoE of remote desktop services for different use cases. For the investigated *typing* use case, the test subjects were asked to type a short text on a text processor in a remote desktop session. The higher the delay in the network, the longer the user has to wait until his actions, e.g., typing a character or deleting character, appear on the screen. The delay until the actions result in visual feedback is denoted as response time and we choose it as the KPI for remote terminal work. Figure 4(a) illustrates the piece-wise interpolated model based on the presented opinion scores in [7]. The authors only investigated response time values up to 0.5 s. We linearly extrapolate the results up to 1.2 s where the utility reaches 1. We define M as $M^{(\text{SSH})}(t) := MOS^{(\text{SSH})}(t) - 1) \cdot \frac{4}{3.3} + 1$ to project the MOS values to a utility range of $[1, 5]$.

4.2.2 Web Browsing / File Downloads. Egger et al. [10] propose models for the user experience of web browsing and file downloads based on subjective user studies. The web browsing model uses the page load time as KPI. For the file download, the download time of a 10 MB file is used as KPI. The MOS value for web browsing is proposed as $MOS^{(\text{WEB})}(t) := -0.88 \cdot ln(t) + 4.72$. For the file download, $MOS^{(\text{DL})}(t) := -1.68 \cdot ln(t) + 9.61$.

Figure 4(b) illustrates the web browsing model. The figure highlights the severe impact of the page load time on the user experience in web browsing. After only 2.2 s waiting time, the MOS is already down from 5 (*Excellent*) to 4 (*Good*). With additional 4.6 s waiting time, the MOS decreases to 3 (*Fair*). After a total waiting time of 20 s, the score ranges between *Poor* and *Bad*. For web downloads (Figure 4(c)), the users are more willing to accept longer waiting times. For

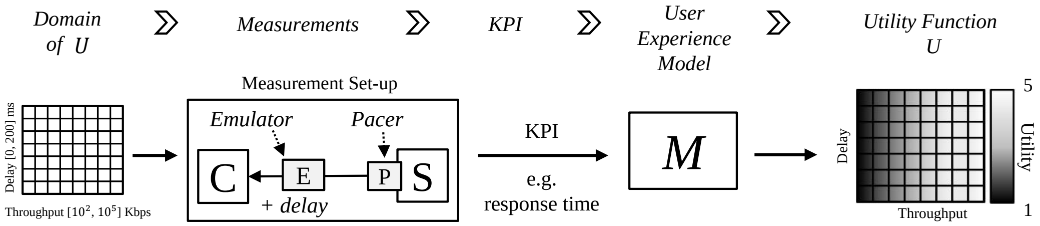


Fig. 5. Utility functions for *(class, intent)* are generated by first defining a measurement domain in terms of throughput and delay. Second, the domain is quantized and the application KPIs are measured in an emulated network environment using the quantized parameters for throughput and delay. Third, user experience models are used to derive the utility for the measured parameters.

example it takes a waiting time of 28 s for the opinion score to decrease to 4. We use the $MOS^{(\mathrm{DL})}$ model as proposed by the authors as M with $M^{(\mathrm{DL})}(t) := MOS^{(\mathrm{DL})}(t)$. $M^{(\mathrm{WEB})}$ we define as $M^{(\mathrm{WEB})}(t) := (MOS^{(\mathrm{WEB})}(t) - 1) \cdot \frac{4}{3.6} + 1$.

4.2.3 Adaptive Video Streaming. The user experience during an adaptive video streaming session depends on factors such as average presented quality, number and amplitude of quality switches, frequency and duration of stalling events, device's screen size, viewing environment, user expectation, encoding, adaptation strategy, and content type [24]. To the best of our knowledge there is no holistic model for the user experience of adaptive streaming available at the moment. One option for enterprises is to create custom models, for example for onboarding videos for new employees.

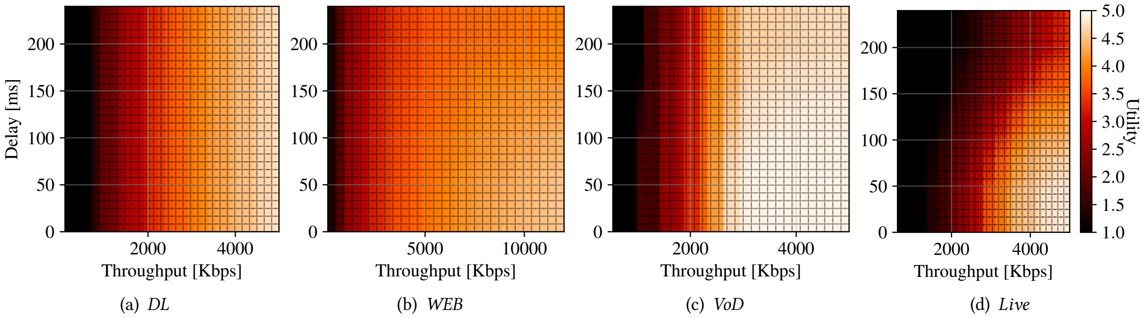
Studies show the average quality as a dominant influence factor [25] for the user QoE. We therefore assign a utility value to a streaming application based on the observed average quality $q^{(\mathrm{avg})}$ and the maximum and minimum quality level, $q^{(\mathrm{max})}$ and $q^{(\mathrm{min})}$. The utility value is then determined by $M^{(\mathrm{HAS})}(q^{(\mathrm{avg})}) := \frac{q^{(\mathrm{avg})} - q^{(\mathrm{min})}}{q^{(\mathrm{max})} - q^{(\mathrm{min})}} \cdot 4 + 1$.

4.2.4 Voice-over-IP. Sun et al. [39] propose a model for the MOS of VoIP depending on the used audio codec and a user's interactivity, i.e., whether the user is only listening or also conferencing. The MOS value is presented as polynomial equation with constants a to j and with packet loss ratio and delay as input parameters. The constants depend on the used codec. We configure D-ITG to emulate G.729. The MOS model $MOS^{(\mathrm{VoIP})}(\mathrm{loss}, \mathrm{delay})$ is then described by Eq. 10 and Table II in [39]. We define the M accordingly as $M^{(\mathrm{VoIP})}(\mathrm{loss}, \mathrm{delay}) := MOS^{(\mathrm{VoIP})}(\mathrm{loss}, \mathrm{delay}) - 1) \cdot \frac{4}{2.65} + 1$.

4.3 Utility Functions

The utility function U_a: (Throughput [Kbps], Delay [ms]) $\mapsto [1, 5]$ approximates the QoE-aware utility for a specific application type a for a unidirectional pacing rates and maximum delay threshold using the utility model. Hence, the function solves the problem of linking network resource demands with the resulting user experience. The hereinafter described methodology for constructing the utility functions can be applied in an automated fashion to any enterprise application and its intents.

Figure 5 illustrates the process of constructing the utility functions. A set-up measures the utility of each application and intent for different pacing rates and delays in an isolated environment. Two hosts (Host S and Host C) are connected through a network emulator. On the emulator, Linux *netem* is adding delay to all packets passing through it. Host S is running the server endpoint of the application, e.g., in case of web browsing an HTTP web server. The client endpoint is assigned to Host C, e.g. the web browser. Host S egress traffic is paced using the *cfg* queuing discipline (Section 6.2). From the measurements we derive the 2-dimensional utility functions. Note that to account for asymmetric data-rates in a conversation, which is the case for the most server-client traffic such as web traffic, the two directions



(a) *DL* (b) *WEB* (c) *VoD* (d) *Live*

Fig. 6. Utility functions U_a: (Throughput, Delay) $\mapsto [1, 5]$ which map throughput and delay to utility for the application classes file download, web browsing and video streaming and intents defined in Table 4.

of a conversation have to be described by different utility functions. For the sake of simplicity, we consider only one direction per conversation as constrained and only present the server-to-client utility functions. For the throughput, we measure *DL* in the range of $[100, 5000]$ Kbps, *WEB* in the range of $[100, 12000]$ Kbps, *VoD* and *Live* in the range of $[750, 5000]$ Kbps and *VoIP* and *SSH* in the range of $[100, 500]$ Kbps. For the delay, we measure *WEB*, *DL*, *VoD*, *Live* in the range of $[0, 240]$ ms and *VoIP* and *SSH* in the range of $[0, 500]$ ms. The maximum pacing rate per intent is set so that further increasing the pacing rate does not improve the utility for any delay demand.

Figure 6 presents the measurement results for the utility of the applications depending on delay and throughput. The intersections of the grid indicate the quantization as used by the resource allocation problem formulation. The figure shows that *DL*, *WEB*, and *VoD* are highly dependent on the throughput and only a minor dependency on delay is visible. *Live* depends on delay and throughput. *SSH* (not shown) depends solely on the delay. For *DL* (Fig. 6(a)), the impact of the delay is limited to the TCP handshake, the file request and acknowledgements packets. The impact is insignificant compared to the download time and not visible on the figure. For *VoD*, the impact of delay depends additionally on the number and playtime duration of video segments and the adaptation strategy. As illustrated by Figure 6(c), the influence of delay for the intent *VoD* is minor. For *Live* there is a clear influence of delay on the utility (Fig. 6(d)). For *SSH*, the delay is the important influence factor, as every typed character triggers an outgoing packet and requires an immediate response packet. As we use persistent HTTP connections for web browsing, there is no influence of the delay on the *WEB* utility due to the TCP handshake. The influence of the delay is limited to the requests of the HTML index object and the embedded resources (Fig. 6(b)).

The maximum utility values an application can reach in the measurements are determined by implementation-specific factors and the domain and range of the utility function. For example *WEB* is limited by the browser processing time and *VoD*/*Live* depend on the behavior of the adaptation algorithm. *SSH* can reach the highest utility of 5 with 100 Kbps throughput and 0 ms delay. *VoIP* can reach 5 with 100.0 Kbps and 34.5 ms. For *WEB* the highest utility is 4.5 with 11589.7 Kbps and 33.1 ms delay. *VoD* can reach its highest utility of 4.9 with 3479.3 Kbps and 41.4 ms. *Live* can reach 4.8 with 5000.0 Kbps and 41.4 ms. *DL* can reach a utility of 4.8 with 5000 Kbps and 99.3 ms delay.

5 UTILITY ALLOCATION PROBLEM

The network controller performs the calculation of the shares to be allocated based on the number of applications, their utility functions, the network topology, current network status, and fairness criteria. For this paper we define the utility fairness criteria as follows. We first try to maximize the minimum utility over all applications (*max-min-fairness*) and afterwards maximize the sum of utilities while allowing a small decrease in minimum utility. The complete allocation formulation is introduced in Appendix A.

We formulate the problem as a Mixed Integer Linear Program (MILP). The objective of the MILP in the first step is to maximize the minimal utility value $\theta^{(\text{min})}$ over all applications. In the second step the MILP maximizes the sum of all utility values, while the minimum utility $\theta^{(\text{min},2)}$ is restricted to the range $\theta^{(\text{min},2)} \in [(\theta^{(\text{min})}1 - \epsilon), \theta^{(\text{min},1)}]$ with $\epsilon = 0.3$. The MILP has to consider the two-dimensional utility function of every application, the capacities of all paths between application endpoints, and the delay at intermediate hops depending on the link utilization. The decision variables describe which pacing rate to apply to which application and how to configure the routing between application endpoints.

We allocate a specific data-rate per application. Hence, we do not consider how much data-rate is actually consumed by an application. On the one side, static allocation via application pacing can guarantee predictable application performance as this work shows in the experiments. But on the other side, there is no statistical multiplex gain in case the applications use less resources than allocated to them. As a consequence, the network may be under-provisioned and available resources are potentially not made available to other applications.

In this work we configure all applications in the experiments to constantly use the link which makes the number of active applications equal to the total number of applications on a given link. That puts the most stress on the link for a given number and mix of applications. Reducing the activity of an application would be equivalent to reducing the number of simultaneously active applications on the link. But existing research on Internet traffic and congestion can be leveraged by future work, e.g., to overprovision the links based on the actual number of active applications at a given point in time.

5.1 First Step: Maximize Minimum Utility

$\mathcal{A}, a \in \mathcal{A}$ is the set of all unidirectional application flows a. We define $\Lambda(a)$ as the target utility value of an application flow a. In the first step we maximize the minimum utility value (*max-min fairness*) subject to all application utilities have to be larger than the minimum utility value $\theta^{(\text{min})}$.

$$\begin{aligned} \text{maximize:} \quad & \theta^{(\text{min})} \\ \text{subject to:} \quad & \Lambda(a) \geq \theta^{(\text{min})} \quad \forall a \in \mathcal{A} \\ & \text{and (7) - (21) in appendix A2 - A6.} \end{aligned}$$

We denote the optimal value of $\theta^{(\text{min})}$ of the first step as $\theta^{(\text{min},1)}$. A full definition of all symbols is provided in the appendix A.

5.2 Second Step: Maximize Sum of Utilities For Constrained Minimum Utility

In the second step we relax the max-min constraint by ϵ and maximize the sum of all target utility values. We add the additional constraint to bound $\theta^{(\text{min})}$ by $\theta^{(\text{min},1)} - \epsilon = 0.3$:

$$\begin{aligned} \text{maximize:} \quad & \sum_{a \in \mathcal{A}} \Lambda(a) \\ \text{subject to:} \quad & \theta^{(\min)} \geq \theta^{(\min,1)} - \epsilon \\ & \text{and (7) - (21) in appendix A2 - A6.} \end{aligned}$$

For the remainder of the paper, if not otherwise stated, $\theta^{(\min)}$ denotes the optimal value of the second step ($\theta^{(\min,2)}$). The complete formulation of the problem can be found in Appendix A.

6 EXPERIMENT DESIGN AND SET-UP

The objective of the experiments is to show the dependability and scalability of resource allocation via end-host pacing and how the different application classes profit and/or suffer from the enforced packet pacing. The experiments are conducted in a set-up where we monitor sets of increasing number of parallel applications sharing a throughput-constrained link. For each set of applications we measure the utility with and without resource allocation and discuss the differences in the evaluation. Dynamic embedding of applications at run-time and additional intents are out of scope of this evaluation. Next, we elaborate on the deployed experimental set-up (Section 6.1) and the custom pacing implementation (Section 6.2). Afterwards, we discuss the experiment parameters (Section 6.3). The results of the evaluation are presented in the subsequent Section 7.

6.1 Experiment Set-up

Figure 7 illustrates the experiment set-up, consisting of two groups of hosts: one server (①) and one client group (②). The link between the two groups is throughput-constrained and the applications running on the host groups have to share the limited bandwidth. The network consists of two switches, one SDN-enabled Pica8 P-3290 (③) and one unmanaged off-the-shelf 100 Mbps switch (④). The link between the two switches constrains the available data-rate between the hosts on the left and on the right side to 100 Mbps. The Pica8 switch is equipped with a maximum queue size of 1 MB and maximum queuing delay of about 80 ms towards the 100 Mbps link. We deploy three modern desktop PCs on each side to meet the processing and memory resources required by the experiment scenarios.

Each application consists of a server and client endpoint, e.g., a web server and a browser. All endpoints are confined to a separate network namespace (⑤) and connected via virtual interfaces and a software bridge to the host's physical interface (⑥). Each namespace is configured with a unique IP and MAC address. Furthermore, every client is connected to an exclusive server application. That way, the pacing rate can be set per namespace and no further control is needed to assign outgoing server packets to different pacers. In case of web browsing, video streaming, and web download, each client is assigned to an exclusive light-weight HTTP server, but with shared content. The server endpoints are placed left of the bottleneck and the client endpoints to the right of the bottleneck, which makes the egress queue and interface of the Pica8 the bottleneck. Pacers ($\boxed{\text{P}}$) based on our *cfq* implementation (Section 6.2) restrict the egress rate of the namespaces/applications towards the hosts' software bridges.

All management and monitoring operations are performed out-of-band. The KPIs of each application are measured at the client endpoint, e.g., the page load time at the browser, and reported to the network controller by the applications' agents (⑦). Additionally, we frequently poll the statistics counters of all physical and virtual network interfaces to measure throughput, queue length and packet loss.

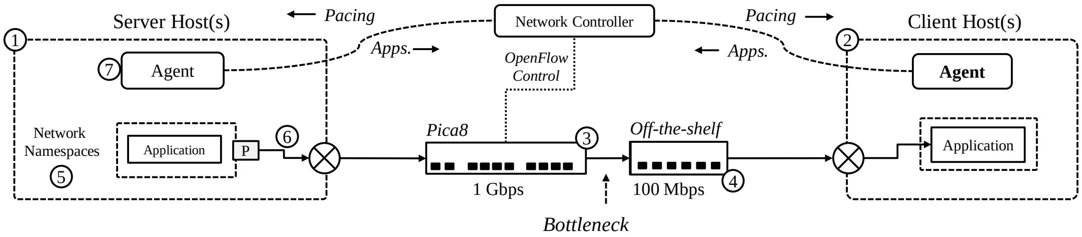


Fig. 7. Experimental set-up. Two groups of hosts, one server and one client group, are connected via an SDN-capable switch and an unmanaged 100 Mbps link to each other. A network controller calculates fair shares, configures the pacers, and collects statistics.

6.2 Pacing Implementation

In Linux, pacing is implemented as a *queuing discipline*. Furthermore, a mechanism called TCP small queues[8] exerts backpressure on the applications to mitigate buffer bloat and packet loss by limiting the allowed number of Bytes per flow in the queuing discipline and device queue (default: 128 Kilobytes). Other operation systems offer similar pacing mechanisms. We implemented a custom queuing discipline based on the existing Fair Queuing (*fq*) discipline [9], referred to as Custom Fair Queuing (*cfq*). Every conversation defined by *(class, intent)* and by one or multiple sockets, can be assigned to an exclusive queue with a target packet release rate as configured by the network controller through the local agent. Packets from the queues are released time-based. The departure time of the next packet $time\_next\_packet$ is determined by the current time now, the size of the current packet pkt_len and the target pacing rate $target\_rate$: $time\_next\_packet = now + \frac{pkt\_len}{target\_rate}$

6.3 Parameter Space and Experiment Procedure

The parameter space of the experiments is limited to the *number* and *types* of the applications and whether the experiment is *managed* or *best effort*. In detail, the bottleneck link is shared by $\{2, 4, .., 24\}$ applications per class, in total $|\mathcal{A}| \in \{10, .., 120\}$. For video streaming, half of the applications are of type *Live* and the other half of *VoD*.

At the start of the experiment, a configuration file is pushed to each host telling the host which number and type of applications to start. Each application is modified to start in its own network namespace and to report its type to the host-local agent (⑤ in Fig. 7). The local agent forwards this information to the network controller. Once all applications are registered with the network controller, the controller calculates the resource shares of utility for each application and pushes the corresponding static pacing rates to the agents. The agents configure the pacers of the applications' network namespaces accordingly. The pacing rate is not changed during an experiment run. The SDN-enabled Pica8 switch is configured via OpenFlow for simple forwarding. Besides the forwarding rule configuration, the OpenFlow connection is used to poll queue and interface statistics.

The duration of one experiment run is 15 minutes with an additional 1 minute warm-up and cool-down phase. The applications are started at random times during the warm-up phase and requests during the warm-up or cool-down phase are discarded for the evaluation. Each experiment is repeated 11 times. If an application's request is finished, it initiates a new request after a pause time of 100 ms. One request equals one video view for *VoD* and *Live*. For *VoIP*, one request equals one 30 s phone call. The reason for the static pause time of 100 ms is that this results in an almost constant number of concurrent applications using the bottleneck link. Hence, each application in a specific scenario is

[8] https://lwn.net/Articles/506237/, Accessed: 2018-10-12
[9] https://lwn.net/Articles/564825/, Accessed: 2018-10-12

constantly sending/receiving requests/responses, except of a 100 ms break between requests to allow for a reset of an application's state. Increasing the pause time between requests would effectively decrease the number of concurrently active applications at a specific point in time. *Cubic* is configured as TCP congestion control algorithm. Cubis is chosen as comparison as it shows better performance on congested links compared to Compound and New Reno TCP [1] and it is the default algorithm for many Linux server variants. BBR congestion control proposed by Google fails to show performance benefits and fairness in heterogeneous environments [23] compared to Cubic.

There exist valid optimal solutions to the allocation problem formulation with applications of the same type to be assigned different utility values. For easier presentation of the results, we constrain the problem formulation to choose one utility value per type. The bottleneck link is modeled with a capacity of 100 Mbps. As the sum of all paced flow rates does not exceed the available capacity, and due to the short pause times between application requests, the link in the managed case is slightly under-provisioned. Thus, a large queue build-up is unlikely and the link delay of the bottleneck is modeled with a constant delay of 2 ms. In the best effort case, the link is already over-utilized with 10 competing applications and experiences 58 ms delay and 0.5 % packet loss (discussed later in Section 7.3).

We provide details on how the experiment setup is expressed in the terms of the variables of the theoretical problem formulation in Appendix B.

7 EVALUATION

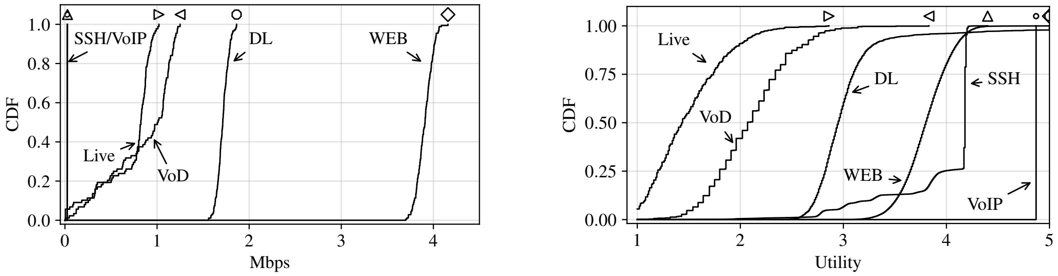
We evaluate the performance of an increasing number of applications sharing a throughput-constrained link with and without data-rate management. The evaluation is pursuing the following questions. i) How does the minimum and average utility of the applications compare between the managed and best effort scenarios? ii) Which applications benefit, which utility values are decreased, and why? iii) Can pacing result in configurable and thus predictable application performance in terms of the difference between the target and the measured utility? iv) How fair, in terms of utility, are the best effort and the managed utility distribution?

First, we evaluate how the available data-rate is distributed among the applications in a best effort scenario and present the resulting utility distribution. Second, we solve the allocation formulation for the scenario, implement static pacing in the set-up for each application and present the gains in terms of utility. Third, we present how pacing affects the QoS parameters, such as packet loss and jitter, of the link. Fourth, we conduct a parameter study on the number of parallel applications and show how the gains and fairness changes with increasing number of parallel applications. Error bars in the result figures indicate the standard deviation if not otherwise stated. In cases the error bars are not clearly visible on the presented scale, they are omitted from the figures.

7.1 Best Effort Throughput and Utility Distribution

First, we take a close look at the best effort application performance for single scenario with 16 clients per application class, 16 × *WEB*, 16 × *DL*, 16 × *SSH*, 16 × *VoIP*, 8 × *VoD*, 8 × *Live*, in total $|\mathcal{A}| = 80$. The scenario with 80 applications is selected for a closer inspection due to the fact that among the investigated application counts (from 10 up to 120 applications), one of the highest gains is observed here. With the 80 applications competing for the bandwidth, the link is fully utilized resulting in an average packet loss of 4 % and queuing delay of 80 ms.

Figure 8(a) presents the CDFs of the average throughput and Figure 8(b) the CDFs of the utility values of all requests per application type. Multiple observations can be made from the figures. First, the throughput as well as the utility is distributed non-uniformly between the application types. For example, while *WEB* enjoys high throughput and utility (median ≥ 3.9 Mbps, 3.8 utility), *Live*'s achieved throughput is less than 1 Mbps and median utility is about 1.4. *WEB*'s



(a) Throughput

(b) Utility

Fig. 8. Best effort throughput and utility of the different application types for 16 clients per application class. The markers at the top are for better visual indication of the application types.

high throughput is due to the use of multiple persistent parallel TCP connections, while video streaming clients, *DL*, and *SSH* establish only one TCP connection. Parallel TCP connections allow an application to receive a proportional larger fraction of the available throughput. As web download has no idle periods during the download, web download exhibits a higher average throughput than video streaming.

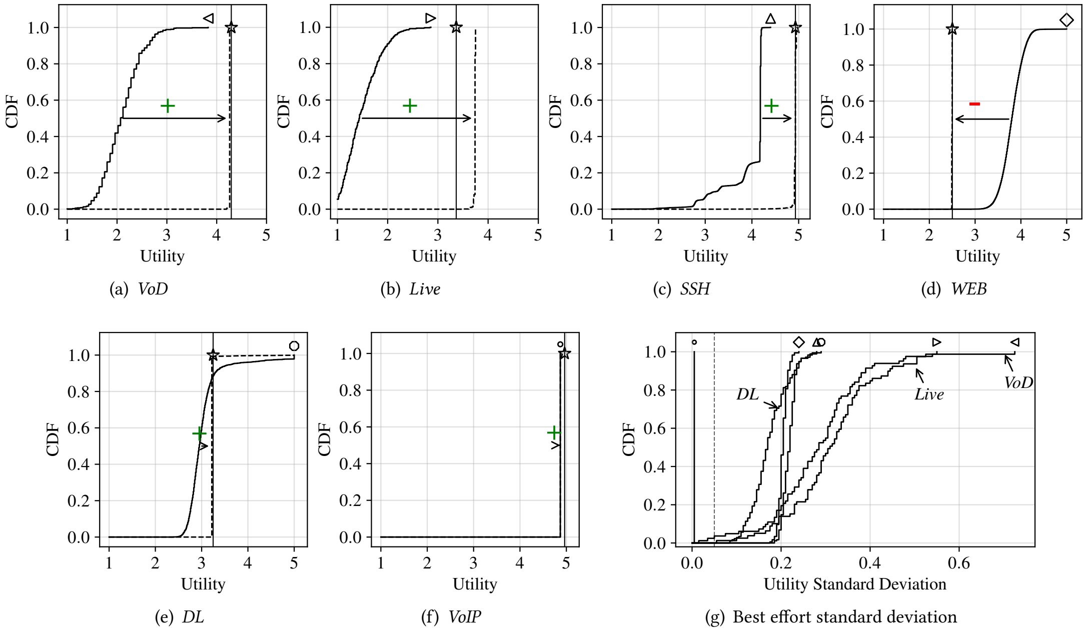
Second, even *VoD* and *Live*, which belong to the same application class (video streaming) and achieve similar throughput rates, suffer from unfair utility distribution (1.3 vs. 2.1). This is due to the smaller playback buffer for live streaming and the increased encoding overhead for the shorter video chunks. Third, the average throughput of *SSH* and Voice-over-IP (*VoIP*) is below 100 Kbps, while the utility is 3.7 and 4.9, respectively. *SSH*'s performance is influenced by delay, caused by queuing at the bottleneck link, and retransmissions, due to lost packets when the bottleneck's queue is overflowing. *VoIP* is barely influenced in this scenario, as the maximum delay and packet loss over the single bottleneck is acceptable for *VoIP* traffic according to the user experience model. Details on the performance of *VoIP* is given in Section 7.3. Fourth, the utility distributions per application type are varying with a standard deviation of 0.2 (*WEB*) to 0.5 (*DL*), with the exception of *VoIP*. Hence, application performance is not consistent across requests of the same application type, and, as a consequence, there is an unfair distribution of shares, even within the same application type.

In summary, best effort delivery is inadequate to provide fair and consistent application performance for multiple applications sharing a constrained link. Best effort delivery does not consider different demands (throughput vs. delay-sensitivity), transport protocols (TCP vs. UDP), or multiple flows per application. Furthermore, the constrained link is overloaded, resulting in lost packets and queuing delay.

7.2 Managed Utility Distribution

Next, we solve the allocation problem formulation with the max-min fairness criteria for the scenario with 80 parallel applications and apply the calculated pacing rates. Figures 9(a) to 9(f) illustrate the best effort (solid lines) and managed utility (dashed lines) for the scenario with 16 clients per application class. Improvements in median utility due to the data-rate management are indicated by (→, +). Deteriorations are shown by (←, −). The target utility per application type, as calculated by the allocation formulation, is indicated by (|, ★).

The figures (a) to (f) show that all application types, except *WEB* and *VoIP*, profit from the management. *Live* benefits most from the management, with a median increase of 3.1 (from 1.3 to 4.4). *VoD*, *SSH* and *DL*'s median utility improve by 2.0, 1.0, and 0.4, respectively. On the other hand, *WEB*'s median utility decreases by 1.3 (from 3.8 to 2.5). With



(a) *VoD*

(b) *Live*

(c) *SSH*

(d) *WEB*

(e) *DL*

(f) *VoIP*

(g) Best effort standard deviation

Fig. 9. Figures (a) to (f) show measured best effort and managed application utility for $|\mathcal{A}| = 80$ applications sharing the constrained link. The dashed lines indicate the utility CDF for the managed scenario, the solid lines the best effort scenario. The star and vertical line mark the target utility Λ for the application type. Arrows to the right highlight the improvement in median utility. Figure (g) show the standard deviations of a client's utility values per application type.

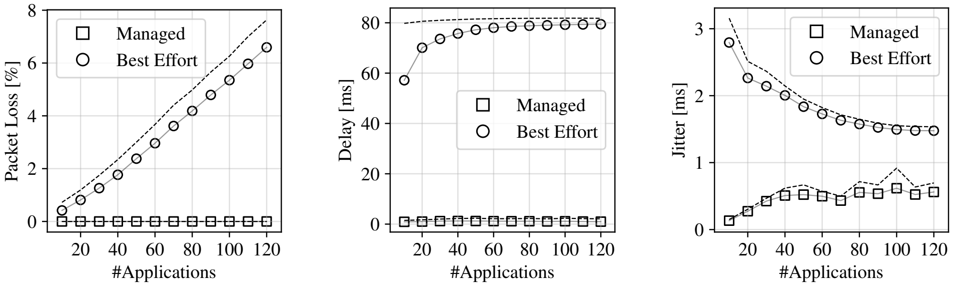
pacing *WEB* can not get an unfair advantage over the other applications by using multiple parallel TCP connections. No noteworthy improvement or deterioration in utility is measurable for *VoIP*.

Live (b) exhibits a deviation of about 0.5 between the target and measured utility. The deviation is the result of an inaccuracy in the live streaming utility function. The samples collected from the utility measurement setup are supplemented with interpolated values to build the quantized utility function. In the case of live streaming and low delay values, the interpolation results in a utility error of about 0.5. The error can be reduced by collecting more measurement samples from the throughput-delay parameter space and/or fine-tuning the interpolation algorithm.

Figure 9(g) presents the *standard deviations per client* of a specific type for the *best effort* scenario. The smaller the standard deviation is, the more consistent is the experience of a single user. The dashed vertical line indicates the maximum (= 0.05) of the standard deviations in the managed case (per type CDFs are not shown for the managed case). *DL* (◯) clients exhibit the largest median standard variation (0.64) among the application types, followed by *SSH* (△) with 0.41. *WEB* (◇) clients' median variation is the second smallest with 0.25. There is no visible variation for *VoIP* (∘). The figure also shows that not only the utility value per client request varies, but also the behavior of each client. For example for *VoD* (◁), the standard deviation varies between 0.1 and 0.43. Hence, some clients experience a smaller quality variation for their video views than other clients.

7.3 Link QoS and VoIP Performance Details

Next, we take a closer look at the QoS metrics of the constrained link in terms of packet loss, queuing delay and jitter for an increasing number of parallel applications. In the best effort case we expect the link QoS parameters to degrade



(a) Packet Loss (b) RTT (c) Jitter

Fig. 10. Quality of Service metrics of the constrained link in terms of packet loss, delay and jitter for increasing number of applications ($|\mathcal{A}|$) as recorded by the VoIP clients. The dashed lines without markers indicate the 95th percentile. The markers indicate the managed (□) and best effort (○) median values. Without data-rate management the queue at the bottleneck is overflowing quickly even at low numbers of parallel applications and thus causing packet loss and delay.

because the link is fully saturated and the interface queue is overflowing. In the managed case we do not expect any degradation as the level of link saturation is managed. As the MOS and utility functions of *VoIP* are based on the QoS metrics, we also discuss why the QoS metrics have only minor influence on the *VoIP* performance in the evaluation.

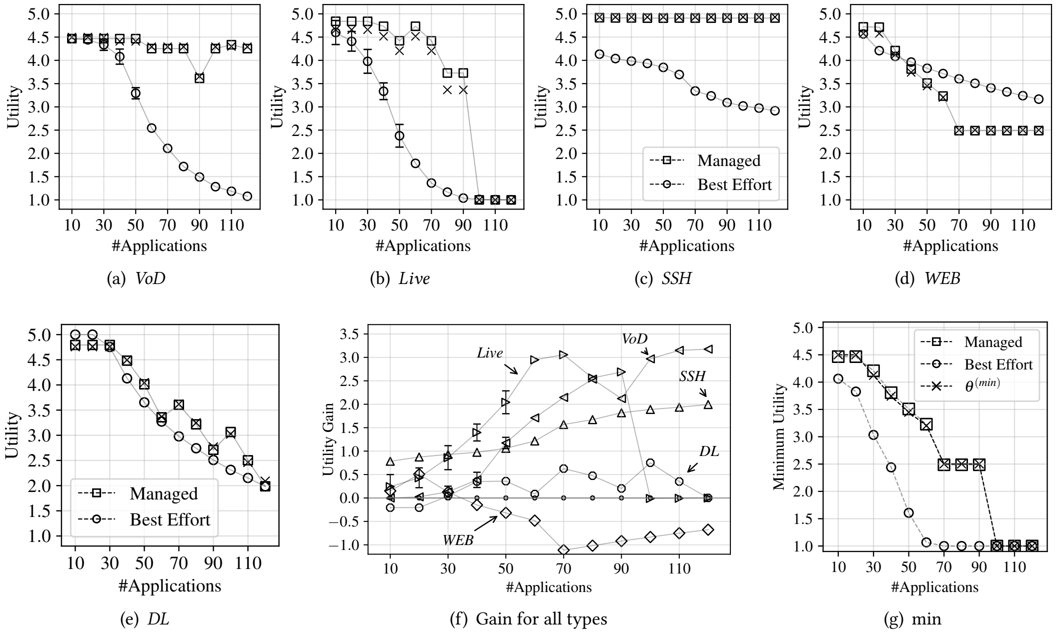
Figure 10(a) shows the median packet loss as measured by the VoIP clients during a call for 10 to 120 parallel applications. The dashed lines indicate the 95th percentile. The figure shows that there is no packet loss for the investigated number of applications in the managed experiments. In the best effort experiments, the packet loss increases linearly from 0.5 % to 7.1 % (0.9 % to 8.1 % for the 95th percentile).

Figure 10(b) shows the Round-Trip-Time (RTT). Note that the client-to-server flow direction of the constrained link is only lightly utilized and therefore, the given RTT approximates the one-way delay experienced by the applications. In the best effort case, the delay increases roughly logarithmic from 53 ms for 10 applications and saturates for 70 parallel applications at 79 ms. The 95th percentile shows that even with 10 parallel applications the experienced RTT is in 5 % of the cases already greater than 79 ms. In the managed experiments the measured RTT increases linearly from 0.9 ms to 1.1 ms (1.5 ms to 2.4 ms).

Figure 10(c) shows the median and 95th percentile of the average jitter as measured by the VoIP clients during a call. In general, the figure shows that in the best effort case the jitter decreases for increasing application count, while for the managed experiments the jitter increases. The decrease in jitter in the best effort case shows that due to the link saturation, there are almost constant inter-arrival times of packets. The high link utilization results in a full link queue and packets are processed at line-rate by the switch's outgoing interface. In the managed case, the arrivals of the multiplexed requests of the clients result in minor RTT variations, but even for 120 applications the 95th percentile of the jitter stays below 0.9 ms.

As there are no retransmissions for VoIP, the maximum delay for the successful transmission of a voice sample is about 80 ms in our set-up. For 8 % packet loss and 80 ms delay, the utility for VoIP is estimated as 4.9 ($U_{VOIP}(80, 0.08) = 4.9$). Hence, as defined by utility function U_{VOIP}, there is a maximum utility difference of 0.1 in the set-up (5 - 4.9).

In summary, data-rate management significantly improves the QoS metrics of the constrained link. There is no packet loss, the RTT stays in most cases far below 2.5 ms and the jitter is at least halved. Regarding the influence of the QoS metrics on the VoIP utility, the VoIP clients in combination with the selected audio codec are marginally affected



(a) *VoD*
(b) *Live*
(c) *SSH*
(d) *WEB*
(e) *DL*
(f) Gain for all types
(g) min

Fig. 11. Comparison of managed measurements (□), target utility (×) and best effort (○) measurements per application type for increasing number of applications sharing the constrained link. In Figure (g), the crosses and the dashed line indicate the solution to the allocation formulation ($\theta^{(\mathrm{min})}$) over all types. Figure (f) summarizes differences in measured utility between best effort and managed. Results are shown over mean of the 10 % tail of all requests of an application.

by the unmanaged link degradation. However, one can imagine how applications with stricter QoS requirements or VoIP calls with longer network paths profit from the QoS improvements.

7.4 Increasing Number of Applications

Figure 11 illustrates the gain in utility per application type for increasing number of simultaneous applications. Results are shown as the mean of the 10th percentiles of the utility values over all requests of an application. The 10 % tail as summary metric is chosen to allow for a small budget of random error compared to the minimal utility over all requests, e.g., for random delays in processing on the experiment PCs or requests which take longer due to rare latency spikes in the network. Hence, on average 90 % of the requests of a client result in a utility equal or better than the given value.

Figures (a) to (e) present the findings per application type. The application class VoIP is omitted as there is no significant difference between the managed and best effort scenario. Figure 11(f) summarizes the difference in utility per application type between the managed and best effort experiments. Application types with a positive difference (top half of the figure) profit from management. The performance of application types with negative differences deteriorate. The following general observations can be made based on the figures.

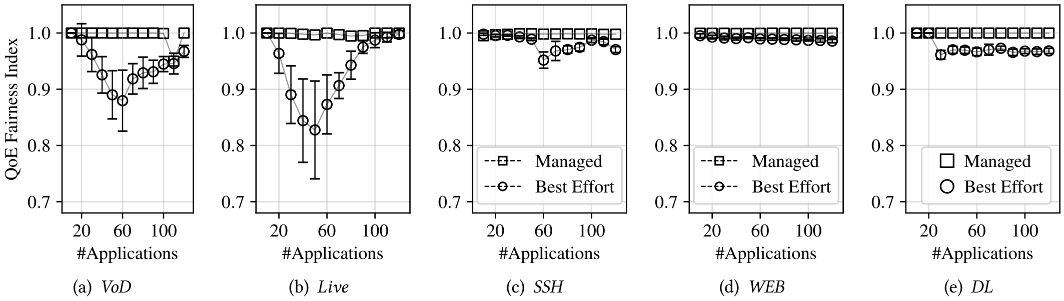
First, the utility for all shown types decreases with increasing number of applications in the best effort case. This is expected as with increasing $|\mathcal{A}|$ more flows compete for the scarce constrained link capacity. In the managed case, only *DL* and *WEB* exhibit an equivalent degradation in utility. *VoD*, *Live*, and *SSH* on the other hand can sustain a high utility in the managed experiments even while the number of competing flows increases. Second, for $|\mathcal{A}| < 40$ the potential gain is low as the available capacity is sufficient to reach close to maximum utility for all applications in the managed and best effort cases. Third, the performance of *WEB* deteriorates while all other classes (except *VoIP*) profit for most of the evaluated values of $|\mathcal{A}|$. Fourth, the minimum utility over all applications ($\theta^{(\min)}$) in the managed case is mostly determined by *WEB* and *Live*. The minimum for the best effort case is mostly dictated by *SSH* for $|\mathcal{A}| < 30$ and by *Live* for $|\mathcal{A}| \geq 30$. Fifth, the measurements from the managed scenario deviate less than 0.5 from the target utility as determined by the solution to the optimization formulation for all application types. For *DL*, *WEB*, *VoD*, and *SSH* the deviation is even less than 0.2 for the investigated number of parallel applications. Hence, data-rate management leads to predictability of application performance. Furthermore, the results show that pacing can implement the output of the allocation optimization formulation accurately.

Next, we investigate the measurement results for each application type in detail. For ***VoD*** (Fig. 11(a)), the utility decreases approximately linear with an increasing number of parallel applications for $|\mathcal{A}| > 30$. For $|\mathcal{A}| \leq 30$, best effort management is sufficient to provide a utility of 4.5 or higher. With data-rate management, the fairness formulation can allocate enough resources to the *VoD* clients to sustain a high utility value even for up to $|\mathcal{A}| = 120$. Hence, for $|\mathcal{A}| = 120$ the utility gain is about 3.1. For ***Live*** (Fig. 11(b)), the figure shows that the utility decreases rapidly without data-rate management. There, data-rate management is most effective at 60 to 70 parallel applications where the increase is up to 3.4. In terms of predictable performance, the target utility is met most of the time with a deviation of 0.1 to 0.3. However for $|\mathcal{A}| \geq 100$, the fairness formulation decreases the utility target to the minimum of 1.0, which is the same low utility as *Live* reaches in the best effort case for the same number of applications. For ***SSH*** (Fig. 11(c)), profit increases roughly linear with $|\mathcal{A}|$, from about 0.7 up to 2.1 for $|\mathcal{A}| = 120$. Data-rate management avoids bursts and keeps the total data-rate under the constrained link capacity. Hence, there is little queuing delay and the delay-sensitive applications like *SSH* can sustain a high utility even for large $|\mathcal{A}|$.

For ***WEB*** (Fig. 11(d)), the difference between managed and best effort is 1 or less utility (maximum difference of 0.9 at $|\mathcal{A}| = 90$). The target utility is close to the measured managed utility. For $|\mathcal{A}| < 90$ and $|\mathcal{A}| > 90$ the difference decreases. As our pacing applies on application level, not flow level, *WEB* can not gain an unfair advantage by opening multiple TCP connections anymore. Furthermore, the utility function of *WEB* (Fig. 6(b)) shows that *WEB* is expensive in terms of required throughput, which makes the optimization likely to sacrifice the target utility of *WEB* in the second optimization step in order to increase the average utility of all applications. ***DL*** (Fig. 11(e)) exhibits the smallest utility gains (besides *VoIP*). The gain is below 0.8 for $|\mathcal{A}| \leq 90$ and around zero for $|\mathcal{A}| = 100$. The decrease of utility with increasing $|\mathcal{A}|$ is roughly linear for the managed and best effort experiments. For $|\mathcal{A}| \geq 100$, the solution to the fairness problem increases the utility target for *DL* again, which results in a utility gain close to 1.0. Managing the utility is accurate and the deviation from the target utility can be neglected for all investigated numbers of parallel applications. ***VoIP*** exhibits no benefit or degradation from the activated management according to the user experience model (further discussed in Section 7.3).

Figure 11(g) shows the minimum 10th percentile utility as measured in the best effort and managed experiments and as calculated by the fairness formulation. The figure shows that in the managed scenario, every client's utility is at least 3.0 up to 80 parallel applications, which is denoted as *fair* on the MOS scale. In the best effort case, the observed minimum utility drops below 3 for 40 applications and down to 1.0 for 80. When comparing $\theta^{(\min)}$ (×) and managed (□), the managed minimum utility does not differ more than 0.1 from the calculated minimum utility.

In summary, the presented measurements for increasing number of parallel applications sharing the constrained link highlight the benefits of the proposed approach. *VoD*, *Live*, *DL*, and *SSH* exhibit gains in utility between 0.5 and up to 3.3, even for 100 and more applications sharing the 100 Mbps link. *WEB*'s utility degrades, but the decrease is less than 1.0. The minimum utility $\theta^{(\min)}$ can be greatly increased, especially for $|\mathcal{A}| > 30$, and the target utility is mostly met,



(a) *VoD* (b) *Live* (c) *SSH* (d) *WEB* (e) *DL*

Fig. 12. Comparison of the F-index between managed (□) and best effort (○) scenarios per application type for increasing number of applications sharing the constrained link. An F-index of 1.0 denotes perfect fairness.

resulting in predictable application performance. *VoIP* shows no benefit or degradation due to the nature of its user experience model.

7.5 QoE Fairness

To the best of our knowledge, there is no fairness measure to quantify the fairness for different application types with orthogonal resource demands, e.g., throughput-sensitive and delay-sensitive demands. For example, *VoIP* is in our set-up always close to a utility of 5.0, independent of other applications. Hence, any fairness measure which considers only differences between values will consider this as unfair. But enforcing equal utility for all application types, including artificially restricting *VoIP*, would result in a non-Pareto-optimal utility distribution where the target utility of *VoIP* could be increased without negatively impacting other applications. Therefore, we evaluate the inter-application fairness per application type. Note that for the evaluation we are restricting the allocation formulation to allocate only one target utility value per application type. Hence the target utilities per type exhibit always perfect fairness and are omitted.

We evaluate the inter-application fairness using the F-index [26] defined by $F = 1 - \frac{2\sigma}{4}$ for a utility scale of 1 to 5. The F-index is selected as fairness measure as it is specifically designed and evaluated for user experience fairness. An F-index of 1.0 indicates perfect fairness between the applications. An F-index of 0.0 is the result of half of the application experiencing a utility of 1.0 and the other half a utility of 5.0. Figures 12(a) to 12(e) illustrate the F-index per application type t for $|\mathcal{A}| = \{10, 20, .., 120\}$ applications sharing the constrained link, measured for the best effort and managed scenarios. From the figures, we conclude that in the managed case, the F-index does not drop below 0.98 for any of the evaluated scenarios and application types.

In the best effort case, the fairness depends strongly on the application type and the number of parallel applications. The *WEB* clients exhibit a fairness similar to that in the managed scenario (≥ 0.98). For *SSH* and *DL*, the fairness fluctuations are larger, but in general the fairness is still high (≥ 0.95). The two video streaming types *VoD* and *Live* suffer the most in the best effort scenarios. For *Live*, the fairness drops down to 0.7 for $|\mathcal{A}| = 44$ and for *VoD* down to 0.77 for $|\mathcal{A}| = 55$. However, for video streaming there is a high level of fairness for $|\mathcal{A}| < 30$ and $|\mathcal{A}| > 100$. This is due to the fact that for low number of parallel applications, there is sufficient capacity for all clients to reach close to maximum utility while for a high number of parallel applications all clients are close to a utility value of 1.0.

In summary, the evaluation of the fairness per application type shows that *VoD* and *Live* profit the most from the management. *SSH* and *DL* show some improvement. *WEB* and *VoIP* improve only marginally. In the managed measurements, we observe nearly perfect fairness for all application types.

7.6 Summary

The evaluation set out to discuss the following four subjects: i) comparison of minimum and average utility for managed and best effort scenarios, ii) advantages and disadvantages of central data-rate management for each application class, iii) predictability of application performance, and iv) fairness between the applications.

First, a scenario with 80 applications sharing a 100 Mbps link is presented. The measurements show that for the best effort case, web browsing consumes about four times more of the available throughput than the other applications. This is due to web browsers using multiple parallel TCP connections. As a consequence, the utility of the web browsing sessions is high (3.5 to 4.0), while other applications like live video streaming suffer ($\leq$ 2). Next, the allocation formulation is solved for the 80 applications and pacing is applied to the applications. The results show that video streaming, remote terminal work, and file download can increase their utility by 1 to 3 while web's utility is only decreased by 1. Furthermore, the standard deviation of a client's utility is decreased to ≤ 0.1 from 0.2 to 0.8 in the best effort case, resulting in predictable application performance. The measurements for 10 to 120 parallel applications sharing the link support the findings of the 80 applications scenario. The evaluation of the fairness shows that in the managed scenarios, the application types exhibit close to perfect fairness. For the best effort scenarios, the fairness results show that the two video streaming intents profit the most from the management, followed by *DL* and *SSH*. The *WEB* clients do not profit much from the management in terms of fairness.

No or little benefit can be expected from the management when the link is only lightly utilized, as the applications do not have to compete for resources and there is no queuing time at the bottleneck. For highly utilized links, throughput-sensitive applications can not profit as the available resources are insufficient for all applications. In such situations some application could be evicted from the network to provide a satisfying experience for critical applications. However, delay-sensitive applications like *SSH* still profit from the reduced queuing at the link.

In summary, the results show that there is a significant benefit of centrally controlled application pacing in terms of utility, inter-application fairness, and predictability. Furthermore, compared to classical Quality of Service measures in the network, the approach can be implemented with heterogeneous forwarding devices without any special features, it does not require expensive switch buffer space, and it is fully software-based.

8 CONCLUSION

In this paper we propose a design for resource allocation in enterprise networks based on central software-defined network control, fine-grained per-application pacing at the end-hosts, and utility functions derived from measurements and user-experience models. Pacing refers to the method of restricting the amount of data an application is allowed to send into the network by implementing local back-pressure to the application sockets and introducing artificial delays between packets. Traditional methods of QoS control in the network, such as policing or scheduling, interact badly with end-host congestion control and do not scale to larger number of applications and application classes. Moving application pacing from in-network QoS methods to the end hosts, e.g., to user PCs, servers, smartphones, and tablets, is scalable, increases transmission efficiency, reduces the required complexity of forwarding devices, and allows cost-efficient high link utilizations. To the best of our knowledge, this is the first work proposing, formulating, and evaluating a scalable architecture for resource allocation for end-user applications in enterprise environments based on real applications and user-experience models.

We define application- and user-level utility using selected user-experience models from the literature. Based on the models, we derive per-application utility models for the five common network use cases web browsing, file download, remote terminal, adaptive video streaming, and Voice-over-IP. Afterwards, we determine sensible resource allocations by formulating a two-stage mixed-integer linear program based on the number and types of applications, their utility functions, and network resources. The mixed-integer linear program decides on how to embed the applications in the network in terms of the allowed data-rate per application and the delay-constrained routing of the application flows. Once the allowed rate and routing is determined, the flow routing is configured through an SDN protocol and the pacing is enforced through local agents at the end-hosts.

We evaluate the methodology by implementing a proof-of-concept testbed with a throughput-constrained link and an increasing number of parallel applications sharing the link. The results show that QoS metrics, such as delay and packet loss, considerably improve with pacing, due to the controlled link utilization. When looking at the fairness per application type, the results show that there is near perfect fairness between the clients. For the five evaluated application types, the results show that web browsing's utility decreases, as it has an unfair advantage in the best effort case due to its multiple parallel TCP-connections. However, the loss in utility of web browsing is low, compared to the gain for the other types. Real-time applications, such as remote terminal work, profit due to the reduced delay and packet loss. VoIP enjoys lower packet loss, delay, and jitter, but does not suffer in terms of utility by one impaired link due to the resilient audio codec. From the experiments, we conclude that the proposed architecture enables scalable resource allocation and predictable application performance.

This paper is a step towards extending Software-defined Networking towards the edge of the network with scalable resource allocations from the perspective of the human users. Future work in this area should focus on how to autonomously create and update utility functions, investigate the impact of inaccurate utility functions, develop fast heuristics for the allocation problem formulation, evaluate further application types, and solve the problem of dynamically recomputing pacing rates and embedding additional applications at run-time.

ACKNOWLEDGMENTS

This work has been supported, in part, by the German Research Foundation (DFG) under the grant numbers KE1863/6-1, ZI1334/2-1 and TR257/43-1 and in part by the European Union's Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets). This work reflects only the authors' view and the funding agency is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Abdeljaouad, I., Rachidi, H., Fernandes, S., and Karmouch, A. Performance analysis of modern TCP variants: A comparison of Cubic, Compound and New Reno. In *2010 25th Biennial Symposium on Communications* (2010), IEEE, pp. 80–83.

[2] Aggarwal, A., Savage, S., and Anderson, T. Understanding the performance of TCP pacing. In *Proc. of IEEE INFOCOM* (2000), pp. 1157–1165.

[3] Botta, A., Dainotti, A., and Pescapè, A. A tool for the generation of realistic network workload for emerging networking scenarios. *Elsevier Computer Networks 56*, 15 (2012), 3531–3547.

[4] Cai, Y., Hanay, Y. S., and Wolf, T. A Study of the Impact of Network Traffic Pacing from Network and End-User Perspectives. In *Proc. of IEEE International Conference on Computer Communications and Networks (ICCCN)* (2011).

[5] Cai, Y., Jiang, B., Wolf, T., and Gong, W. A Practical On-line Pacing Scheme at Edges of Small Buffer Networks. In *Proc. of IEEE INFOCOM* (2010).

[6] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and Jacobson, V. BBR: Congestion-based congestion control. *ACM Queue 14*, 5 (2016), 50.

[7] Casas, P., Seufert, M., Egger, S., and Schatz, R. Quality of experience in remote virtual desktop services. In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)* (2013).

[8] Cheng, Y., and Cardwell, N. Making Linux TCP Fast. In *Netdev Conference* (2016).

[9] De Cicco, L., Caldaralo, V., Palmisano, V., and Mascolo, S. Tapas: a tool for rapid prototyping of adaptive streaming algorithms. In *Proc. of ACM Workshop on Design, Quality and Deployment of Adaptive Video Streaming* (2014).

[10] Egger, S., Reichl, P., Hossfeld, T., and Schatz, R. Time is bandwidth? Narrowing the gap between subjective time perception and Quality of Experience. In *Proc. of IEEE International Conference on Communications (ICC)* (2012), pp. 1325–1330.

[11] Fei, Z., Xing, C., and Li, N. QoE-driven resource allocation for mobile IP services in wireless network. *Springer Science China Information Sciences 58*, 1 (2015), 1–10.

[12] Ferguson, A. D., Guha, A., Liang, C., Fonseca, R., and Krishnamurthi, S. Participatory networking: an API for application control of SDNs. In *Proc. of ACM SIGCOMM* (2013), vol. 43, pp. 327–338.

[13] Ferguson, A. D., Guha, A., Place, J., Fonseca, R., and Krishnamurthi, S. Participatory Networking. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)* (2012).

[14] Flach, T., Papageorge, P., Terzis, A., Pedrosa, L., Cheng, Y., Karim, T., Katz-Bassett, E., and Govindan, R. An Internet-Wide Analysis of Traffic Policing. In *Proc. of ACM SIGCOMM* (2016), ACM, pp. 468–482.

[15] Floyd, S. TCP and explicit congestion notification. *ACM SIGCOMM Computer Communication Review 24*, 5 (1994), 8–23.

[16] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking 1*, 4 (1993), 397–413.

[17] Floyd, S., and Jacobson, V. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking 3*, 4 (1995), 365–386.

[18] Georgopoulos, P., Elkhatib, Y., Broadbent, M., Mu, M., and Race, N. Towards network-wide QoE fairness using openflow-assisted adaptive video streaming. In *Proc. of ACM SIGCOMM Workshop on Future human-centric Multimedia Networking (FhMN)* (2013), pp. 15–20.

[19] Gharakheili, H. H., Vishwanath, A., and Sivaraman, V. Comparing edge and host traffic pacing in small buffer networks. *Elsevier Computer Networks 77*, C (2015), 103–116.

[20] Ghobadi, M., and Ganjali, Y. TCP pacing in data center networks. In *IEEE 21st Annual Symposium on High-Performance Interconnects* (2013), IEEE, pp. 25–32.

[21] Gómez, G., Lorca, J., García, R., and Pérez, Q. Towards a QoE-driven resource control in LTE and LTE-A networks. *Journal of Computer Networks and Communications*, Article ID 505910 (2013).

[22] Hassas Yeganeh, S., and Ganjali, Y. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks* (2012), ACM, pp. 19–24.

[23] Hock, M., Bless, R., and Zitterbart, M. Experimental evaluation of bbr congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)* (2017), IEEE.

[24] Hossfeld, T., Seufert, M., Sieber, C., and Zinner, T. Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming. In *Proc. of Sixth International Workshop on Quality of Multimedia Experience (QoMEX)* (2014), pp. 111–116.

[25] Hossfeld, T., Seufert, M., Sieber, C., Zinner, T., and Tran-Gia, P. Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies. *ELSEVIER Computer Networks 81* (2015), 320–332.

[26] Hossfeld, T., Skorin-Kapov, L., Heegaard, P. E., and Varela, M. Definition of QoE fairness in shared systems. *IEEE Communications Letters 21*, 1 (2017), 184–187.

[27] Hu, T. C. Multi-commodity network flows. *Operations research 11*, 3 (1963), 344–360.

[28] Kumar, A., Jain, S., Naik, U., Raghuraman, A., Kasinadhuni, N., Zermeno, E. C., Gunn, C. S., Ai, J., Carlin, B., Amarandei-Stavila, M., and Others. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In *Proc. of ACM SIGCOMM* (2015), vol. 45.

[29] Li, Z., Zhu, X., Gahm, J., Pan, R., Hu, H., Begen, A. C., and Oran, D. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications 32*, 4 (2014), 719 – 733.

[30] Liu, F., Xiang, W., Zhang, Y., Zheng, K., and Zhao, H. A Novel QoE-Based Carrier Scheduling Scheme in LTE-Advanced Networks with Multi-Service. In *Proc. of IEEE Vehicular Technology Conference (VTC Fall)* (2012), IEEE.

[31] Lukaseder, T., Bradatsch, L., Erb, B., Van Der Heijden, R. W., and Kargl, F. A Comparison of TCP Congestion Control Algorithms in 10G Networks. In *Proc. of IEEE Local Computer Networks (LCN)* (2016), pp. 706–714.

[32] Mirchev, A. Survey of Concepts for QoS improvements via SDN. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM) 33* (2015), 1.

[33] Mittal, R., Dukkipati, N., Blem, E., Wassel, H., Ghobadi, M., Vahdat, A., Wang, Y., Wetherall, D., Zats, D., and Others. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proc. of ACM SIGCOMM* (2015), vol. 45, ACM, pp. 537–550.

[34] Nichols, K., and Jacobson, V. Controlling queue delay. *Communications of the ACM 55*, 7 (2012), 42–50.

[35] Ryu, S., Rump, C., and Qiao, C. Advances in Active Queue Management (AQM) Based TCP Congestion Control. *Springer Telecommunication Systems 25*, 3-4 (2004), 317–351.

[36] Sacchi, C., Granelli, F., and Schlegel, C. A QoE-oriented strategy for OFDMA radio resource allocation based on min-MOS maximization. *IEEE Communications Letters 15*, 5 (2011), 494–496.

[37] Saeed, A., Dukkipati, N., Valancius, V., Contavalli, C., Vahdat, A., and Others. Carousel: Scalable Traffic Shaping at End Hosts. In *Proc. of ACM SIGCOMM* (2017), ACM, pp. 404–417.

[38] Salles, R. M., and Barria, J. A. Fair and efficient dynamic bandwidth allocation for multi-application networks. *Computer Networks 49*, 6 (2005), 856–877.

[39] Sun, L., and Ifeachor, E. C. Voice quality prediction models and their application in VoIP networks. *IEEE Trans. on Multimedia 8*, 4 (2006), 809–820.

[40] Tang, P., Wang, P., Wang, N., and Ngoc, V. N. QoE-Based Resource Allocation Algorithm for Multi-Applications in Downlink LTE Systems. In *Proc. of International Conference on Computer, Communications and Information Technology (CCIT)* (2014), Atlantis Press, pp. 1011–1016.

[41] Tootoonchian, A., and Ganjali, Y. Hyperflow: A distributed control plane for OpenFlow. In *Proceedings of the 2010 internet network management*

conference on Research on enterprise networking (2010), vol. 3.

[42] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612.

[43] Wei, D., Cao, P., Low, S., and EAS, C. TCP Pacing Revisited. In *Proc. of IEEE INFOCOM* (2006).

[44] Wurtzler, M. Analysis and simulation of weighted random early detection (WRED) queues.

A ALLOCATION PROBLEM FORMULATION

Next we give the complete description of the resource allocation problem formulated as a MILP. The MILP has to consider the two-dimensional utility function of every application, the capacities of all links and the delay on intermediate links depending on the link utilization. The decision variables describe which pacing rate to apply to which application and how to configure the routing between application endpoints. The problem can be summarized with the following inputs, objectives, high-level constraints and outputs.

Inputs: (I) Number of applications. (II) Utility function U of each application. (III) Network topology with link capacity information and delay on the links based on link utilization.

Objectives: (I) Min-max utility fairness in the first step. (II) Increasing average utility in the second step.

Constraints: Unidirectional application routing (source to destination) has to be valid, considering link capacity and maximum delay per application.

Outputs: (I) Target utility value and allocated throughput per application. (II) Application flow routing.

A.1 Notation

Table 5 summarizes the notation. $\mathcal{A}, a \in \mathcal{A}$ is the set of all unidirectional application flows a. For simplification, application flow a and intent i are merged in the notation to only a and each application consists of only one application flow. The two directions of a bidirectional application flow are considered as two independent applications by the formulation. This allows different paths and utility functions for both flow directions. We define the topology as a directed graph $G(\mathcal{V}, \mathcal{E})$ with nodes $v \in \mathcal{V}$ and edges $(u, v) \in \mathcal{E}$ and edge capacity $C_{u,v}$. A flow a is defined by the source node S_a, target node T_a and its utility function U.

ψ and Ψ (both $\in \mathbb{R}_+{}^{|\mathcal{E}|\times|\mathcal{E}|\times m}$) describe the piece-wise defined relationship between link usage ψ and delay Ψ for specific edge and for a quantization bin m.

The utility function describes the relationship between allocated throughput and delay and the application's resulting utility (Fig. 6). It can be determined for example through measurements and user experience models, as we do in the paper at hand in Chapter 4. Mathematically, the utility function is split into its three components, the throughput demands ($\tau \in \mathbb{R}_+{}^{|\mathcal{A}|\times n}$), the delay demands ($\delta \in \mathbb{R}_+{}^{|\mathcal{A}|\times n}$) and the utility values ($U \in ([1, 5])^{|\mathcal{A}|\times|\tau|\times|\delta|}$), where n denotes the quantization bin.

F describes the application flow routing. An edge (u, v) is traversed by an application a if $F_{a,u,v}$ equals 1. Delay on a link is describes as a function of the link usage. ψ and Ψ (both $\in \mathbb{R}_+{}^{|\mathcal{E}|\times|\mathcal{E}|\times m}$) define the piece-wise defined relationship between usage (ψ) and resulting delay (Ψ) for each edge in the graph G and quantization bin m.

A.2 Objective

The objective of the MILP is in the first step to maximize the minimal utility value $\theta^{(\min)}$ over all applications. In the second step the MILP maximizes the sum of all utility values while the minimum utility is $\theta^{(\min)}$ restricted to range based on the minimum value determined by the first step, denoted as $\theta^{(\min,1)}$, $\theta^{(\min)} \in [\theta^{(\min,1)} - \epsilon, \theta^{(\min,1)}]$ with $\epsilon = 0.3$. The second step allows the problem formulation to improve the average utility over all applications by relaxing the max-min fairness constrain using the slack parameter ϵ. This prevents solutions where the optimization would stop when the utility of a single application can not be increased further, but where there are plenty of resources left to increase the utility of other applications.

Table 5. Notation Allocation Problem Formulation

Symbol	Type	Unit	Description
Constants			
$G(\mathcal{V}, \mathcal{E})$			Network topology graph with nodes $\mathcal{V}$ and edges $(u, v) \in \mathcal{E}$.
$\mathcal{A}, a \in \mathcal{A}$			Set of all unidirectional application flows.
S, T	$\in \mathcal{V}^{\lvert\mathcal{A}\rvert}$		Start and target nodes of application flows.
ψ, Ψ	$\in \mathbb{R}_+{}^{\lvert\mathcal{E}\rvert \times \lvert\mathcal{E}\rvert \times m}$		Translation between link usage and delay for a specific link.
C	$\in \mathbb{R}_+{}^{\lvert\mathcal{V}\rvert \times \lvert\mathcal{E}\rvert}$	Kbps	Unidirectional link capacity between u and v.
τ	$\in \mathbb{R}_+{}^{\lvert\mathcal{A}\rvert \times n}$	Kbps	Utility functions' throughput demands of the applications.
δ	$\in \mathbb{R}_+{}^{\lvert\mathcal{A}\rvert \times n}$	ms	Utility functions' delay demands of the applications.
U	$\in ([1, 5])^{\lvert\mathcal{A}\rvert \times \lvert\tau\rvert \times \lvert\delta\rvert}$		Utility functions' utility values of the applications.
Decision Variables			
$\theta^{(\text{min})}$	$\in [1, 5]$		Minimum utility for all applications.
T	$\in \{0, 1\}^{\lvert\mathcal{A}\rvert \times \lvert\tau\rvert}$		1 if a specific throughput demand index is selected for an application.
Δ	$\in \{0, 1\}^{\lvert\mathcal{A}\rvert \times \lvert\delta\rvert}$		1 if a delay demand index for application is selected.
F	$\in \{0, 1\}^{\lvert\mathcal{A}\rvert \times \lvert\mathcal{E}\rvert \times \lvert\mathcal{E}\rvert}$		1 if an edge is traversed by an application.
Functions			
$\eta(a)$	$\mathcal{A} \mapsto \mathbb{R}_+$	Kbps	Selected throughput for application a.
$D(a)$	$\mathcal{A} \mapsto \mathbb{R}_+$	ms	Selected delay requirement for application a.
$\Lambda(a)$	$\mathcal{A} \mapsto [1, 5]$		Target utility value of application a.
$\Omega(u, v)$	$\mathcal{E} \mapsto \mathbb{R}_+$	Kbps	Assigned throughput to link (u, v) in Kbps.
$\omega(u, v)$	$\mathcal{E} \mapsto \mathbb{R}_+$	ms	Delay on link (u, v) in milliseconds.
$\Upsilon(a)$	$\mathcal{A} \mapsto \mathbb{R}_+$	ms	End-to-end delay of application a in milliseconds.
Miscellaneous			
$\theta^{(\text{min},\{1\|2\})}$	$\in [1, 5]$		Solution of $\theta^{(\text{min})}$ in first and second step.
ϵ [= 0.3]	$\in \mathbb{R}^+$		Slack parameter for $\theta^{(\text{min})}$ in the second step.
n, m	$\in \mathbb{N}$		Quantification factors for the utility and link delay functions.

We define θ_a as utility value of an application a. In the first step we maximize the minimum utility value (*max-min fairness*) subject to all application utilities have to be larger than the minimum utility value $\theta^{(\text{min})}$:

$$\text{maximize:} \quad \theta^{(\text{min})} \tag{1}$$

$$\text{subject to:} \quad \Lambda(a) \geq \theta^{(\text{min})} \quad \forall a \in \mathcal{A} \tag{2}$$

$$\text{and (7) - (21)} \tag{3}$$

We denote the optimal value of $\theta^{(\text{min})}$ of the first step as $\theta^{(\text{min},1)}$. In the second step we relax the max-min constraint by ϵ and maximize the sum of all utility values. We denote the optimal value of $\theta^{(\text{min})}$ of the second step as $\theta^{(\text{min},2)}$ and add the additional constraint to bound $\theta^{(\text{min},2)}$ by $\theta^{(\text{min},1)} - \epsilon = 0.3$:

$$\text{maximize:} \quad \sum_{a \in \mathcal{A}} \Lambda(a) \tag{4}$$

$$\text{subject to:} \quad \theta^{(\min)} \geq \theta^{(\min,1)} - \epsilon \tag{5}$$

$$\text{and (7) - (21)} \tag{6}$$

For remainder of this formulation and if not otherwise stated, $\theta^{(\min)}$ denotes the optimal value as determined by the second step ($\theta^{(\min,2)}$). Next we formulate the constraints. Table 6 summarizes the constraints.

A.3 Utility Selection Constraints

For each application, one throughput, delay and target utility value have to be selected. We first introduce the equations and afterwards illustrate the selection process by a simplified example. Eq. 7 and Eq. 8 dictate that only one throughput and delay demand for application a can be chosen at a time:

$$\sum_{i=1}^{|\mathrm{T}_a|} \mathrm{T}_{a,i} = 1 \quad \forall a \in \mathcal{A} \tag{7}$$

$$\sum_{i=1}^{|\Delta_a|} \Delta_{a,i} = 1 \quad \forall a \in \mathcal{A} \tag{8}$$

Hence the chosen throughput demand in Kbps η^a and delay requirement in milliseconds D^a for application a are given by the following element-wise multiplications.

$$\eta(a) := \mathrm{T}_a^T \cdot \tau_a \tag{9}$$

$$D(a) := \Delta_a^T \cdot \delta_a \tag{10}$$

The resulting utility value of application a, $\Lambda(a)$, is then selected from the quantified utility functions (Fig. 6) by the following equation:

Table 6. Overview of all constraints

Type	Constraints	Description
Objectives	(2), (5)	Maximize minimum utility (1st step) and sum of utilities (2nd step).
Utility	(7) - (11)	Select target utility, throughput allocation and maximum allowed delay per application.
Routing	(12) - (14)	Application routing (multi-commodity flow problem).
Capacity	(15) - (16)	Link capacity (in Kpbs) can not be exceeded by applications.
Delay	(17) - (21)	Determine delay per link (in milliseconds) depending on link usage. Ensure applications' maximum delay demand is not exceeded.

$$\Lambda(a) := \sum_{tp=1}^{|\mathrm{T}|} \sum_{d=1}^{|\Delta|} (\mathrm{T}_{a,tp} \cdot \Delta_{a,d} \cdot U_{a,tp,d}) \tag{11}$$

Next we give an example for a target utility, throughput and delay demands calculations for an arbitrary application a. The discretized utility function U_a has a domain of [100, 500, 1000] Kbps for the throughput and [150, 100, 50] milliseconds for the delay demand. At an allocation of 1000 Kbps and 50 ms the utility of the application reaches its highest point with 4.9, while for 100 Kbps and 150 ms the target utility drops to 1.3. In the following example the decision variables $\mathrm{T}_{a,1}$ and $\Delta_{a,1}$ are set to 1 by the solver based on other constraints like the available link capacity. Hence, an allocation of $\eta(a) = 500$ Kbit/s is chosen with a target utility of $\Lambda(a) = 3.0$.

$$\Lambda(a) = \begin{array}{cc} & \begin{array}{ccc} \mathrm{T}_{a,0} & \mathrm{T}_{a,1} & \mathrm{T}_{a,2} \end{array} \\ \begin{array}{c} \Delta_{a,0} \\ \Delta_{a,1} \\ \Delta_{a,2} \end{array} & \begin{pmatrix} U_{a,0,0} & U_{a,1,0} & U_{a,2,0} \\ U_{a,0,1} & U_{a,1,1} & U_{a,2,1} \\ U_{a,0,2} & U_{a,1,2} & U_{a,2,2} \end{pmatrix} \end{array} = \begin{array}{cc} & \begin{array}{ccc} 0 & \mathbf{1} & 0 \end{array} \\ \begin{array}{c} 0 \\ \mathbf{1} \\ 0 \end{array} & \begin{pmatrix} 1.3 & 1.6 & 2.1 \\ 2.9 & \mathbf{3.0} & 3.5 \\ 4.2 & 4.3 & 4.9 \end{pmatrix} \end{array} = 3.0$$

$$\eta(a) = \begin{pmatrix} \tau_{a,0} \\ \tau_{a,1} \\ \tau_{a,2} \end{pmatrix} \cdot \begin{pmatrix} \mathrm{T}_{a,0} & \mathrm{T}_{a,1} & \mathrm{T}_{a,2} \end{pmatrix} = \begin{pmatrix} 100 \\ \mathbf{500} \\ 1000 \end{pmatrix} \cdot \begin{pmatrix} 0 & \mathbf{1} & 0 \end{pmatrix} = 500\,\mathrm{Kbit/s}$$

$$D(a) = \begin{pmatrix} \delta_{a,0} \\ \delta_{a,1} \\ \delta_{a,2} \end{pmatrix} \cdot \begin{pmatrix} \Delta_{a,0} & \Delta_{a,1} & \Delta_{a,2} \end{pmatrix} = \begin{pmatrix} 150 \\ \mathbf{100} \\ 50 \end{pmatrix} \cdot \begin{pmatrix} 0 & \mathbf{1} & 0 \end{pmatrix} = 100\,\mathrm{ms}$$

A.4 Routing Constraints

We formulate the application flow routing problem as the multi-commodity flow problem [27] with non-fractional flows. First we formulate the constraints required to route the flow from source to destination. Afterwards we formulate the link capacity and application delay constraints. A flow is subject to the following routing constraints. Number of incoming and outgoing edges of in-between nodes has to be equal (*flow conservation*):

$$\sum_{w \in \mathcal{V}} F_{a,u,w} = \sum_{w \in \mathcal{V}} F_{a,w,u} \qquad | \; u \neq T_a, S_a \qquad \forall a \in \mathcal{A}, \forall u \in \mathcal{V} \tag{12}$$

Flow conservation at the source (Eq. 13) and destination (Eq. 14):

$$\sum_{w \in \mathcal{V}} F_{a,S_a,w} - \sum_{w \in \mathcal{V}} F_{a,w,S_a} = 1 \quad \forall a \in \mathcal{A} \tag{13}$$

$$\sum_{w \in \mathcal{V}} F_{a,w,T_a} - \sum_{w \in \mathcal{V}} F_{a,T_a,w} = 1 \quad \forall a \in \mathcal{A} \tag{14}$$

A.5 Capacity Constraints

Capacity constraints ensure that the assigned throughput to a link does not exceed the capacity of the link. Next we formulate the required link capacity constraints. We define the link usage in Kbps $\Omega(u, v)$ on the directed edge (u, v) as the sum of the throughput values of all applications traversing that edge/link:

$$\Omega(u,v) := \sum_{a \in \mathcal{A}} F_{a,u,v} \cdot \eta(a) \tag{15}$$

And assigned throughput can not exceed the capacity:

$$\Omega(u,v) \leq C_{u,v} \quad \forall (u,v) \in \mathcal{E} \tag{16}$$

A.6 Delay Constraints

We define the delay of each link as a function of the link usage. That way, the delay function can express a combination of constant, e.g., propagation delay, and dynamic, e.g., queuing and processing delay, use cases. For example, an added constant delay can describe significant propagation delay, or the queuing delay can be modeled based on the target link utilization. We first provide the necessary equations and then provide a simple example.

We do a piece-wise linear interpolation to approximate the link delay for edge (u,v), denoted as $\omega(u,v)$, for a given link usage $\Omega(u,v)$ of the edge. $\psi_{u,v,i}$ and $\Psi_{u,v,j}$ describe the piece-wise defined translation sets between a usage in Kbps with index i and delay in milliseconds with index j for a link (u,v) with $|\psi_{u,v}| = |\Psi_{u,v}|$. We introduce the variables $l_{u,v,p}$ with $l_{u,v,p} \in \{0,1\}$ and $S^{u,v,p} \in [0,1]$ for $p = \{0,1,..,|\psi_{u,v}|-1\}$. Variable l selects the closest, lower, link usage from ψ and S is the linear scaling factor. l and S are subject to:

$$S_{u,v,p} \leq l_{u,v,p} \quad \forall (u,v) \in \mathcal{E}, \ \ p = \{0,1,..,|\psi_{u,v}|-1\} \tag{17}$$

Constrain the selection variable $l_{u,v,p}$ and scale variable $S_{u,v,p}$ according to the link usage $\Omega_{u,v}$:

$$\Omega(u,v) - \sum_{p=0}^{|\psi_{u,v}|-1} [l_{u,v,p} \cdot \psi_{u,v,p} + (\psi_{u,v,p+1} - \psi_{u,v,p}) \cdot S_{u,v,p}] = 0 \quad \forall (u,v) \in \mathcal{E} \tag{18}$$

$\omega(u,v)$ then defines the delay for the given link usage:

$$\omega(u,v) := \sum_{p=0}^{|\psi_{u,v}|-1} [l_{u,v,p} \cdot \Psi_{u,v,p} + (\Psi_{u,v,p+1} - \Psi_{u,v,p}) \cdot S_{u,v,p}] \tag{19}$$

Let's consider the following simple example. A hypothetical link (u,v) has a maximum capacity of 1000 Kbps and a propagation delay of 10 ms. Up to a link usage of 100 Kbps, there is no queuing delay. Between 100 Kbps and 1000 Kbps the queuing delay increases linearly up to a maximum of 70 ms. Hence, at a link usage of 1000 Kbps the delay on the link is 70 ms + 10 ms = 80 ms. We can model this by setting ψ and Ψ as follows:

$$\psi_{u,v} = \begin{pmatrix} \psi_{u,v,0} \\ \psi_{u,v,1} \\ \psi_{u,v,2} \end{pmatrix} = \begin{pmatrix} 0 \\ 100 \\ 1000 \end{pmatrix} Kbps \quad \Psi_{u,v} = \begin{pmatrix} \Psi_{u,v,0} \\ \Psi_{u,v,1} \\ \Psi_{u,v,2} \end{pmatrix} = \begin{pmatrix} 10 \\ 10 \\ 80 \end{pmatrix} ms$$

Let us assume the decision variables assign link (u,v) a total link usage of 500 Kbps. The resulting total delay on that link can then be calculated by first determining l and S:

$$\Omega(u,v) - \sum_{p=0}^{|\psi_{u,v}|-1} [l_{u,v,p} \cdot \psi_{u,v,p} + (\psi_{u,v,p+1} - \psi_{u,v,p}) \cdot S_{u,v,p}] = 0$$

$$\leftrightarrow 500 - ([l_{u,v,0} \cdot 0 + (100 - 0) \cdot S_{u,v,0}] + [l_{u,v,1} \cdot 100 + (1000 - 100) \cdot S_{u,v,1}]) = 0$$

The statement is true for $l_{u,v} = [0, 1]$ and $S_{u,v} = [0, 0.\bar{4}4]$. The delay on the link is then calculated as follows:

$$\omega(u,v) = \sum_{p=0}^{|\psi_{u,v}|-1} [l_{u,v,p} \cdot \Psi_{u,v,p} + (\Psi_{u,v,p+1} - \Psi_{u,v,p}) \cdot S_{u,v,p}]$$

$$=[l_{u,v,0} \cdot 10 + (10 - 10) \cdot 0] + [1 \cdot 10 + (80 - 10) \cdot 0.\bar{4}4] \approx 41\,\text{ms}$$

The end-to-end delay of an application is then the sum of delays on the links traversed by the application. We denote the end-to-end delay of application a with $\Upsilon(a)$:

$$\Upsilon(a) := \sum_{(u,v)\in\mathcal{E}} \omega(u,v) \cdot F_{a,u,v} \tag{20}$$

Finally, the delay of the flow is not allowed to exceed the requirement:

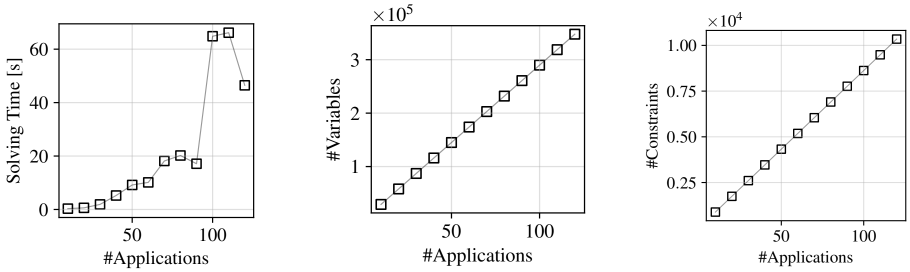
$$\Upsilon(a) \leq D(a) \quad \forall a \in \mathcal{A} \tag{21}$$

A.7 Problem Complexity and Possible Solving Strategies

The optimization formulation combines variations of the non-splittable multi-commodity flow problem (routing) [27] and of the knapsack problem (balancing demand and utility), both known to be NP-hard. Hence, approximation algorithms have to be found to solve the formulation in a reasonable runtime for larger topologies with potentially multiple bottleneck links and a large number of simultaneous applications. The efficient and fast solving of the problem is out of scope of this work and is left to future work. This work provides the necessary abstractions and implementation proof that once the allocation decision is made, it can be efficiently and accurately be implemented in the network. As with other network resource allocation problems, such as the virtual network embedding (VNE) problem, the efficient solving of the theoretical problem can now be explored independently of the implementation concepts.

Solving the problem for our evaluation scenario (one bottleneck link, ≤ 120 applications) takes on average less than one minute on a standard eight-core Intel Core i7-4770 3.4 GHz desktop PC with 32 GB RAM using the commercial Gurobi[10] solver. In detail, Figures 13(a) to 13(c) illustrate the solving time, total number of variables and total number of constraints of the problem instances with increasing number of applications with one bottleneck link. The solving time stays below 10 s up to approximately 50 applications. Above 90 applications the solving time increases drastically up to 66.2 s. Afterwards, when a high number of applications does not leave much room for allocating higher utility values, the solving time decreases again. Figures 13(b) and 13(c) show that the number of variables increases linearly with the number of applications with 2889 variables and 86 constraints for each additional application. Thus, the total number of variables and constraints depends on the number of applications, on the used quantification of the utility and link delay functions and on the size of the network topology.

[10] http://www.gurobi.com/



(a) Solving Time

(b) Variables

(c) Constraints

Fig. 13. Problem size and solving time of the optimization formulation for increasing number of applications ($|\mathcal{A}|$) sharing *one* bottleneck link. Maximum of 66.2 s solving time for 110 applications. 2896 variables and 86 constraints for each additional application.

One greedy algorithm for finding a viable solution could be to start with a target utility of 1.0 for all application flows and shortest path routing. Subsequently the utility can be increased by increments of 0.1 in a round-robin order until an allocation is reached where no application's utility can be increased anymore without violating capacity or delay constraints. One problem with this algorithm is that it does not find sophisticated solutions where the utilization of one path is kept low to support low volume-low delay applications, e.g., web browsing, and other paths are dedicated to batch transfers, e.g., file download.

Despite sophisticated approximations there may be delay between a change to the global state, e.g., a new application, and the availability of a new allocation. Applications may have to wait before they can join the network, lower priority applications have to be disconnected or some throughput has to be reserved for yet unknown applications. This reserved capacity can then be allocated to new applications without requiring the solver to recalculate.

B EXPERIMENT VALUE SETUP

In this section we describe the value setup used in the experiments in this paper for a two applications setup. For increasing number of applications, all variables with dependency on $\mathcal{A}$ increase in size along their first dimension. Table 7 summarizes the problem input variables. In the experiments we have a network topology G with one bidirectional link ($\mathcal{E} = [(0,1),(1,0)]$) between two nodes ($\mathcal{V} := [0,1]$). The link is shared by two application flows ($\mathcal{A} := [0,1]$) which both send data from node 0 to node 1 ($S := [0,0]$, $T := [1,1]$). The link has a capacity of 100 Mbps in both flow directions ($C := [100\ Mbps, 100\ Mbps]$). The link delay is modeled as constant with 2 ms for our managed scenarios where combined paced throughput does not exceed the link capacity ($\psi := [0, 100\ Mbps]$, $\Psi := [2\ ms, 2\ ms]$). τ, δ and U describe the quantized utility functions from Fig. 6. An example for the quantization of the utility functions can be found in Section A.3.

Table 7. Experiment Value Setup For Two Applications

Symbol and value	Description
Problem Input Variables	
$G(\mathcal{V} := [0,1], \mathcal{E} = [(0,1),(1,0)])$	Network topology with two nodes and unidirectional links between them.
$\mathcal{A} := [0,1]$	2 applications competing for the link.
$S := [0,0]$, $T := [1,1]$	Application flows are from node 0 to node 1.
$\psi := [0, 100\ Mbps]$, $\Psi := [2\ ms, 2\ ms]$	Constant delay of 2 ms for link (0,1).
$C := [100\ Mbps, 100\ Mbps]$	The link has speed of 100 Mbps in both directions.
τ, δ, U	Quantized utility functions. See A.3 for an example.