Ralf Kundel, Paul Stiegele, Dat Tran, Julian Zobel, Osama Abboud, Rhaban Hark, Ralf Steinmetz. User Space Packet Schedulers: Towards Rapid Prototyping of Queue-Management Algorithms Accepted for publication at the Conference on Networked Systems (NetSys), 2021.

ECEASST

User Space Packet Schedulers: Towards Rapid Prototyping of Queue-Management Algorithms

Ralf Kundel[1], Paul Stiegele[1], Dat Tran[1], Julian Zobel[1], Osama Abboud[2], Rhaban Hark[1], Ralf Steinmetz[1]

ralf.kundel@kom.tu-darmstadt.de
[1] Multimedia Communications Lab (KOM)
Technical University of Darmstadt, Germany
[2] Huawei Technologies Duesseldorf GmbH, Germany

Abstract: Quality of Service indicators in computer networks reached tremendous importance over the last years. Especially throughput and latency are directly influenced by the dimension of packet queues. Determining the optimal dimension based on the inevitable tradeoff between throughput and latency tends to be a hard, almost infeasible challenge. Several algorithms for *Active Queue Management* have been proposed to address this challenge over the last years. However, the deployment and by that the development of such algorithms is challenging as they are usually located within the operation systems' kernel or implemented in fixed hardware. In this work, we investigate how novel algorithms can be deployed in user space for rapid prototyping with tolerable effort. We provide core performance characteristics and highlight the viability and reasonability of this approach.

Keywords: AQM, User Space, Congestion Control, Bufferbloat

Introduction, Background, and Related Work: Internet applications have experienced an enormous upswing, most notably through intensive video streaming and conferencing applications realized by congestion-controlled TCP. Besides steadily increasing requirements regarding throughput and availability, especially latency became more and more important as a Quality of Service (QoS) criterion. However, there is a tradeoff between high bandwidth utilization and low latency packet forwarding for congestion-controlled flows when determining the dimension of packet queues on forwarding routers.

Throughput-intensive applications require sufficiently large packet queues in order to entirely utilize a link, whereas latency-sensitive applications suffer from too large packet queues, known as *bufferbloat* phenomenon [GN12]. The optimal queue size in terms of maximizing throughput without unnecessary increase in latency is considered to be $B = \frac{\mathrm{RTT} \cdot C}{\sqrt{n}}$, where B is the maximum queue size, C the bottleneck link speed, and RTT the Round Trip Time of n TCP flows [AKM04]. However, this formula and its application on packet queue engineering has several challenges. First, this "optimal" queue size primarily focuses on full utilization of the subsequent bottleneck link and only secondary on latency reduction. Second, constantly having n congestion-controlled TCP flows with a known RTT is not realistic as Internet traffic changes dynamically and numerous different congestion control algorithms are present in parallel [KWM+19]. Third, this rule of thumb considers only long data transmissions, so called "elephant flows", although there is always a mixture of long and short flows in real networks.

Ralf Kundel, Paul Stiegele, Dat Tran, Julian Zobel, Osama Abboud, Rhaban Hark, Ralf Steinmetz. User Space Packet Schedulers: Towards Rapid Prototyping of Queue-Management Algorithms Accepted for publication at the Conference on Networked Systems (NetSys), 2021.

To this end, numerous general but advanced Active Queue Management (AQM) algorithms have been proposed [Ada13][KBV+18]. They tackle the problem of *bufferbloat* by intelligently dropping packets or marking them with a congestion notification bit. However, no algorithm fitting all use cases has been found to this point and it is unlikely to exist. Certainly, the development and experimental evaluation of such algorithms is challenging as they are either located within the operating system kernel, *e.g.*, the Linux kernel, or within hardware of packet forwarding chips (i.e., ASICs).

With this work, we propose the idea of packet queuing and scheduling in the user space for rapid prototyping and preliminary evaluation of novel AQM algorithms. Note that congestion control experiments are usually not performed by simulations, due to an easier reproduction of complexity and heterogeneity of real computer networks in emulation environments, mainly Mininet. Hence, our work focuses on network emulations.

The most related approach to this work is the Linux kernel extension *libnetfilter_queue*[1]. This library enables a hybrid coexistence of queues in the kernel space and a packet scheduler in the user space. The scheduler decides whether a packet is dropped or not based on packet metadata information. However, advanced queuing and scheduling structures exceed the API capabilities, *e.g.*, *front drop* instead of *tail drop* or *Push-In-Extract-Out* queues cannot be investigated.

Prototype and Preliminary Evaluation Results: In order to realize a user space queue emulator, we first investigate which programming languages should be used with regard to their packet I/O performance. For that, we create a simple Mininet topology $h1 \longleftrightarrow h2 \longleftrightarrow h3$ consisting of three hosts. Host h1 and h3 perform a bandwidth performance test with the *IPerf3* tool. Host h2 is responsible to forward all packets from h1 to h3 and vice versa without queuing. To evaluate the performance of the packet forwarding, we measure the throughput of the *IPerf3* TCP flow and the observed round trip time (RTT). Performance characteristics for six different packet forwarding implementations are displayed in Figure 1.

We use native *Linux kernel* and the userspace implementation in *C* as baseline. Forwarding incoming packets based on the *Linux kernel's* IPv4 routing table represents a theoretically achievable upper bound for throughput and a lower bound for RTT. The low-level user space implementation in *C*, compiled with -lpcap flags and no further optimizations, is supposed to be a even better upper bound for throughput. We realized four implementations in different programming languages with similar behavior to the baseline designs for the user space queuing system. Note that the user space implementations are built upon a raw socket kernel module and thus the kernel is still involved. They are implemented as multi-threaded applications and use raw sockets to receive and send packets. One thread receives packets on the first interface and immediately sends them out on the second one. The second thread is responsible for similar packet forwarding in the opposite direction. More threads per direction are only meaningful if a hardware network interface card with load balancer is in use. The third and main thread is responsible to monitor these two workers. This multi-threading approach achieves almost a twofold increase in forwarding performance compared to a single-threaded approach.

As depicted in Figure 1, within the user space implementations, *Go* achieves the best RTT. The *C* user space baseline-implementation, in contrast, achieves the best throughput, but surprisingly also experiences a significantly higher RTT. This is caused by a different interrupt handling

[1] The netfilter.org libnetfilter_queue project. www.netfilter.org/projects/libnetfilter_queue/. Accessed: 2020-10-13.

Ralf Kundel, Paul Stiegele, Dat Tran, Julian Zobel, Osama Abboud, Rhaban Hark, Ralf Steinmetz. User Space Packet Schedulers: Towards Rapid Prototyping of Queue-Management Algorithms Accepted for publication at the Conference on Networked Systems (NetSys), 2021.

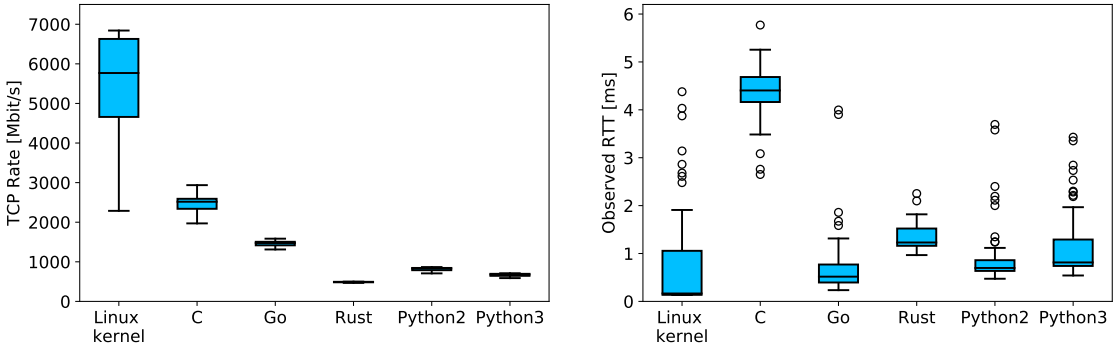
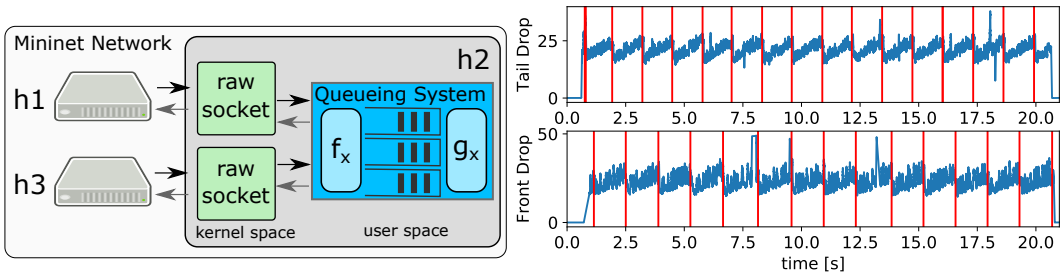


Figure 1: User space packet forwarding characteristics with disabled checksum offloading. The Linux IPv4 forwarding is by default part of the Linux kernel and represents an upper bound. All tests are compiled and running on a Intel Xeon D-1541 with Ubuntu 16.04 (kernel 4.4.0).

structure within the *C* raw socket implementation. Note that this comparison is an evaluation of the raw socket performance within these languages only and not a benchmark of the languages itself. In total, *Go* performed best considering both metrics of the four high-level languages.

As prototyping in the queueing system should be as easy as possible, a high-level language such as *Go*, *Rust* or *Python* is pursued instead of low-level *C*. As it performed best in our preceding tests, we decided to implement the user space queuing system in *Go*, including queues with exchangeable schedulers and AQM algorithms. To showcase the benefits of such a user space queuing system, the prototype was integrated into the previous Mininet topology, as depicted in Figure 2a. For that, two experiments with different queue behavior were performed using (1) a *tail drop* queue and (2) a *front drop* queue with a rate limit of $50 Mbit/s$. In case of a full queue, when packet loss is unavoidable on receiving another packet, the former approach drops the new packet, while the latter drops the first in the queue and then adds the new packet at the tail. As the congestion control detects the packet loss earlier in the latter case, we assume a different behavior in both scenarios which shall be further investigated.

Exemplary results for both approaches are compared in Figure 2b. In this concrete case,



(a) (b)

Figure 2: Results of an exemplary user space queuing experiment. (a) Integration of queueing system into simple Mininet topology. (b) Packet latency for dropping packets in the front/tail of the queue while overflowing. Red bars mark dropped packets (22 tail drops, 15 front drops).

Ralf Kundel, Paul Stiegele, Dat Tran, Julian Zobel, Osama Abboud, Rhaban Hark, Ralf Steinmetz. User Space Packet Schedulers: Towards Rapid Prototyping of Queue-Management Algorithms Accepted for publication at the Conference on Networked Systems (NetSys), 2021.

we observe that the front drop queue creates less packet loss—15 compared to 22 lost packets—because TCP congestion control receives feedback earlier and thus adapts the sending rate sooner. This quickly realized experiment in combination with easily generated but yet expressive results highlight the major advantage of our user space queue emulator. In contrast to kernel or hardware implementations, the user space allows to log all dropping events. In essence, it is easy and convenient to determine why, where, and when a packet is dropped using user space packet schedulers for rapid prototyping.

Conclusion and Outlook: Active Queue Management is highly important in modern networks in order to meet the growing Quality-of-Service requirements. However, their development and evaluation is challenging due to their typical implementation in the kernel or in hardware. We therefore propose the idea of user space queuing systems for rapid prototyping of novel AQM algorithms. We performed experiments comparing four different high level programming languages for their applicability to implement such a system. The results have shown that raw sockets and user space packet queues achieve bandwidths above $1\,Gbit/s$, which is sufficient for most AQM-experiments. Additionally, two simple queuing algorithms were evaluated using a prototypical implementation of our user space queuing system. Because AQM prototyping in user space allows to conveniently test and evaluate novel AQM approaches, future work will focus on the realization of programmable user space queues in real computer networks based on high performance packet I/O frameworks together with high-level programming languages.

Acknowledgements: This work has been supported by the Federal Ministry of Education and Research (BMBF, Germany) within the Software Campus Project ”5G-PCI” and in parts by the German Research Foundation (DFG) as part of the projects B1 and C2 within the Collaborative Research Center (CRC) 1053 MAKI and the LOEWE initiative (Hessen, Germany) within the project Nature 4.0 and the EmergenCity Centre.

Bibliography

[Ada13] R. Adams. Active Queue Management: A Survey. IEEE Communications Surveys Tutorials 15(3):1425–1476, 2013.

[AKM04] G. Appenzeller, I. Keslassy, N. McKeown. Sizing Router Buffers. In Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. P. 281292. ACM, 2004.

[GN12] J. Gettys, K. Nichols. Bufferbloat: dark buffers in the internet. Communications of the ACM 55(1):57–65, 2012.

[KBV+18] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, R. Steinmetz. P4-CoDel: Active Queue Management in Programmable Data Planes. In Proceedings of the Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). Pp. 1–4. 2018.

[KWM+19] R. Kundel, J. Wallerich, W. Maas, L. Nobach, B. Koldehofe, R. Steinmetz. Queueing at the Telco Service Edge: Requirements, Challenges and Opportunities. In Workshop on Buffer Sizing. Stanford, US, 2019.