P4-CoDel: Active Queue Management in Programmable Data Planes

Ralf Kundel, Jeremias Blendin, Tobias Viernickel, Boris Koldehofe, Ralf Steinmetz
Multimedia Communications Lab, Technische Universität Darmstadt, Germany
Email: {ralf.kundel, jeremias.blendin, tobias.viernickel, boris.koldehofe, ralf.steinmetz}@kom.tu-darmstadt.de

***Abstract*—Today, networks are still vulnerable to high latencies. An important reason for that is the Bufferbloat problem caused by big packet buffers as part of the forwarding equipment of modern networks. Even if these buffer sizes are configured in a reasonable way, they still have a fixed size which is always a compromise. Indeed, the required buffer size strongly depends on the RTT of the end-to-end network connection. In order to support dynamic buffer sizes and to reduce the negative impact of Bufferbloat, different Active Queue Management (AQM) algorithms have been proposed recently, such as CoDel and PIE. However, these algorithms are typically not available in state of the art network equipment.**

In this paper we show how recent AQM algorithms can be implemented using P4 programmable network equipment. In consequence, communication networks can be easily enhanced to benefit from state-of-the-art AQM algorithms. To this end, we provide an open-source available implementation of CoDel, one of the most prominent queue management algorithms, in P4. The implementation of such AQMs in P4 data plane hardware enables a massive reduction of latency in many use cases, such as traffic shaping in ISP access networks.

***Index Terms*—P4, Active Queue Management, Bufferbloat, CoDel, Low Latency, Queueing, Traffic Shaping**

I. Introduction

The volume of Internet traffic has risen by an enormous extent in recent decades. Most of the transferred data is transmitted by connection oriented protocols [1], typically TCP. TCP uses a congestion control mechanism [2] in order to avoid over utilization, to ensure fairness, and to provide a high throughput. Whenever packets from a high bandwidth link are forwarded to a link with lower bandwidth congestion can occur. This congestion suffers from the buffering of packets at the egress port of the forwarding switch or router.

Traditionally, the required queue size (B) for a fully utilized and congested link has been advised to be larger than the passing connections' bandwidth-delay product: $B = RTT * C$, with RTT being the round-trip time and C the link capacity [3]. In case of multiple TCP connections at the same time, this demand is reduced to $B = \frac{RTT*C}{\sqrt{n}}$ where n describes the number of congestion controlled connections on the link [4]. Whereas the link speed is mostly constant, the RTT of internet users can vary between less than 10 ms for connections to content delivery-network (CDN) servers and over 200 ms for connections around the world [5]. Thus, the optimal queue size is RTT sensitive. This means that all packet queues with a fixed size are a trade-off between the needs of connections with large and small $RTTs$ in most internet use cases. A queue with constant size, but larger than required, will be filled by packets of the active connections until the maximum window size of TCP is reached or the queue is full and starts dropping packets. This lead to unnecessarily high delays due to full buffers, known as Bufferbloat [6]. These long delays dramatically reduce the quality of latency sensitive network traffic; such as voice over IP (VoIP). However, if queue sizes are too small, this can lead to an underutilization of the congested link.

Thus, the size of a queue should be chosen so that a full utilization of the link leads to a queue utilization which is sometimes close to, but never reaches zero. If the minimum queue utilization is significant larger than zero, the delay increases without any benefit.

Approaches to address the Bufferbloat problem are active queue management (AQM) algorithms like CoDel [7] and PIE [8]. These algorithms do not require an active adjustment of the queue sizes. Instead, they provide an integrated, dynamic feedback mechanism to ensure that queuing delays become as small as possible. The effectiveness of CoDel and PIE has been demonstrated in literature before [9] and they have been included in the Linux kernel. The introduction of Software-defined Networking (SDN) and Network Functions Virtualization (NFV) in many areas of networking, for example in ISP access network infrastructures, led to a new generation of programmable network devices.

Programmable match-action packet-forwarding ASICs [10] give the possibility to implement new functionality without creating new ASICs, which is very costly and time intensive. Most of these ASICs can be programmed with the open-source available programming language P4 [11]. In this paper

- we show that and how it is possible to implement a state-of-the-art AQM algorithm in programmable data planes using P4,
- we provide an open-source implementation, available on GitHub, of the CoDel algorithm in P4.

In Chapter II we consider all fundamental basics and related work. In Section III the implementation of CoDel in P4 is introduced. Chapter IV shows the mininet testbed setup, available on GitHub, and evaluation results.

II. Background

In this chapter we motivate briefly which queueing requirements must be fulfilled for a maximum throughput. Furthermore we outline existing approaches of AQM, introduce programmable data plane hardware and give an overview over related work.

A. Programming Protocol-independent Packet Processors

The high-level data plane programming language P4 [11], introduced in 2014, follows the main objective to define packet processing functionality of programmable network devices. Major benefits are that packet processing of P4 devices can be reconfigured after deployment and are hardware and protocol independent. Thus, P4 circumvents hardware vendor constraints which requires the use of proprietary languages.

The abstract forwarding model of any P4 device starts with a programmable parser, which allows custom defined headers. Next multiple match+action stages, which can be in parallel or in series, may modify headers or determine egress ports. The $P4_{14}$ forwarding model, as shown in Figure 1, has a pipeline, divided into an ingress and egress part. Between those parts, a fixed function block for queueing packets is implemented.

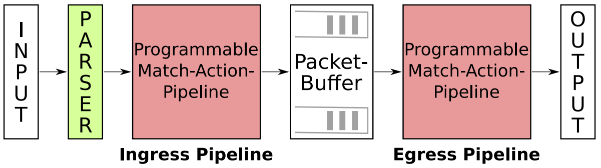
INPUT
PARSER
Programmable Match-Action-Pipeline
Packet-Buffer
Programmable Match-Action-Pipeline
OUTPUT
Ingress Pipeline
Egress Pipeline

Fig. 1. Abstract $P4_{14}$ forwarding model

Currently there are two versions of the language, $P4_{14}$ and the newer version $P4_{16}$. However, since the $P4$ software toolchain for the newest version of the language is not considered production quality yet, we will focus on $P4_{14}$ in this document. When looking at P4 implementations, the abstract forwarding model is constructed by defining header formats, a packet header parser, table and action specifications as well as the control program itself. P4 programs can be executed either in the reference behavioral model (bmv2) or on real hardware like FPGAs, SmartNICs [12] or switches [13].

B. Bufferbloat and Queueing Requirements

For simplicity, we assume an exemplary network with a single router and two ports. On the first port, the packets arrive and are forwarded to the second port of limited bandwidth. The dequeueing rate of the output port, called service rate, is equal to this bandwidth limitation. Figure 2 shows the temporal course of the buffer level for a single TCP connection in such a switch with a simplified congestion control. As soon as the arrival rate exceeds the outgoing service rate, the buffer level will rise until it is full. Then, first packets will be dropped and the congestion control of the TCP connection will detect packet loss. In a consequence, the TCP sender reduces its sending rate and will increase it slowly again afterwards. Since the buffer becomes never empty, the service rate of the queue and the utilization of the bottleneck link is maximized. The red box below the buffer level chart in Figure 2 indicates that there are always packets in the queue. As already mentioned, the queue should be almost empty when the arrival rate exceeds the fixed service rate. If the queue becomes empty before that, the sending rate collapse. In general, TCP tend to fill buffers if no packet loss occurs. If these buffers are sized too big, they create additional queueing delay, called Bufferbloat [6].

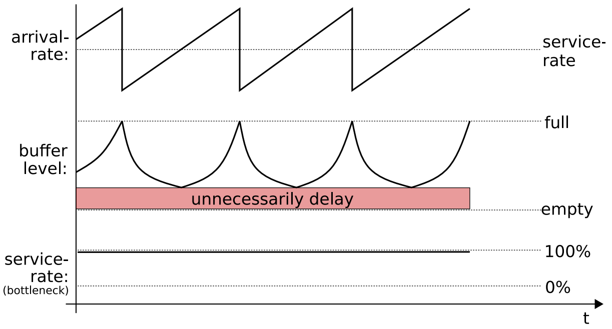

Fig. 2. Buffer level behavior for a single TCP connection

If we consider many TCP connections in parallel, the arrival rate will be a overlay of many TCP saw tooth like courses with a lower total amplitude. By that, the required maximum buffer size will be lower, exactly $B = \frac{RTT*C}{\sqrt{n}}$ [4].

C. CoDel

CoDel (Controlled Delay) is an AQM algorithm that addresses the Bufferbloat problem, specified by the IETF in RFC 8289 [7]. Its effectiveness has been shown in literature before [9]. The pseudocode in Listing 1 shows the main behavior of this algorithm.

```
#define TARGET 5 //ms
#define INTERVAL 100 //ms
Paket p;   StatefulObject s;
if(p.queuedelay < TARGET || q.byte < INRTERFACE_MTU)
  s.dropping = false; count = 0;
  continue;
if(s.dropping == false)
  s.dropping = true;
  s.drop_next_paket = now + INTERVAL;
  continue;
if(s.dropping && s.drop_next_paket >= now)
  drop();
  count++;
  s.drop_next_paket = now + INTERVAL / sqrt(count);
```

Listing 1. Reduced CoDel Pseudocode

The algorithm has two parameters, `TARGET` and `INTERVAL`. The following algorithm ensures that the queueing delay will be periodically lower than `TARGET`:

- If the queueing delay is below `TARGET`, a packet is never dropped.
- If `TARGET` is exceeded by more than `INTERVALL` time units, the first packet will be dropped.
- From now on, the interval between dropping packets is getting smaller, until the `TARGET` delay is reached.

The full algorithm also considers other factors, for example the duration since the last dropping phase. Our implementation is based on the complete algorithm whereas we have shown here a reduced version for better understanding.

D. Related Work

Sharma et.al. have used the possibilities of P4 programmable data planes to establish fairness between multiple

flows by per flow metering and a approximating fair queueing algorithm [14]. A very similar fairness problem was addressed by another work which implemented a equity motivated queueing algorithm for P4 data planes [15].
Sivaraman et.al. analyzed different AQM algorithm and found out that there is "No Silver Bullet" which fits best for all use cases [16]. They have shown that it is possible to implement CoDel with FPGAs, attached to a fixed function forwarding ASIC. Their key insight was, that the data plane must be flexible in order to be open for different AQM algorithms. From their point of view, that could be achieved best by including small FPGAs on switches.

III. System Design

Until recently, state-of-the-art switches, as shown by Sivaraman et.al, cannot be used to implement custom queueing mechanisms. The introduction of programmable data planes, for example with P4, provides new prospects for that. However, P4 is a language that strictly works on and with packet headers, but it doesn't describe the queueing of packets. Still, it can be used to implement CoDel, and thereby an AQM mechanism as we will show.

Our approach is integrated into the P4 reference pipeline, as shown in Figure 3. The packets can be processed in any way by the ingress pipeline. Afterwards they are stored in a queue of the packet-buffering unit in the middle and in the egress pipeline CoDel will control the queueing delay by dropping packets.

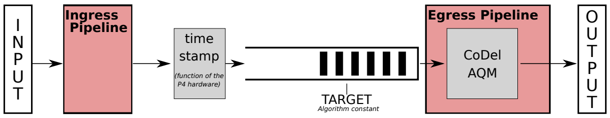

Fig. 3. CoDel integration in P4 reference pipeline

Our open-source available implementation[1] is build for the P4 reference model bmv2 and can be executed on every Linux based system without the need of specialized hardware or proprietary software. Nevertheless, this code can be used as a basis for migrating to P4 compatible hardware. In the egress pipeline the bmv2 provides the possibility to access the queueing delay as packets are timestamped before enqueueing. Considering the CoDel implementation from Listing 1, this is, with expect of the current time, the only required information. Note: the queueing behavior of P4 processors is not standardized and depends on the architecture. However, it is reasonable to assume that on other hardware the same information about the queueing state will be provided in a similar way.
Our implementation is self explainable and can be mapped to provided pseudocode or the rfc reference code with ease. The stateful information, for example the point of time when the next packet should be dropped, are stored in registers. Our implementation can be integrated in any P4 project by integrating the file `codel.p4` and inserting one line at the egress pipeline declaration of the project. For in depth evaluation we provide a second file, `queue_measurement.p4`, which inserts the queueing delay of the packet in the first 32 bit of the TCP payload. For that, a TCP checksum update is required and it can be used only for TCP load tests which don't take care of the payload (f.e. iperf3).

A. Complex Arithmetic Functions in P4

Main difficulty of implementing the CoDel algorithm was the lack of square root function support in P4. With $P4_{16}$ fixed function blocks, which could describe the computation of a square root, are supported. However, an execution on hardware still requires the support of this functionality. Therefore, we introduce an approximation for the function $\frac{INTERVAL}{\sqrt{n}}$ with standard language expressions. If the targeting hardware supports this complex computation it can be replaced easily.

```
...
table_add  t_control_law  action  0/26  =>  17677
table_add  t_control_law  action  0/27  =>  25000
table_add  t_control_law  action  0/28  =>  35355
table_add  t_control_law  action  0/29  =>  50000
table_add  t_control_law  action  0/30  =>  70710
table_add  t_control_law  action  0/31  =>  100000
```

Listing 2. Simple square root approximation

The approximation counts the number of leading zeros by applying a longest prefix match table, as shown in Listing 2. For each entry of the table the result of the frac is stored in there and no complex computation in the data plane is needed. An even better approach is the use of range matching. By that, an approximation can be almost equal to the logarithmic function. However, this will require more memory space than the simple approximation.

B. Other AQMs

According to our design, many AQMs can be described for P4 data planes. One challenge is that some algorithms like PIE [8] decide a packet drop before enqueueing the packets in the ingress pipeline, which has no information about the current queueing delay. For that, a packet can be duplicated in the egress pipeline and its duplicate is sent back by `P4-recirculation` to deliver the current queueing state.

IV. Testbed Setup & Evaluation

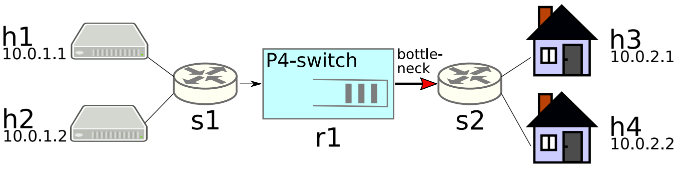

Fig. 4. Mininet testbed setup

Main goal of the evaluation is to show that the implemented algorithm works as expected and by that prove the feasibility of AQMs in P4 data planes. The implementation is embedded in a simple Mininet [17] topology, as shown in Figure 4. In this scenario two Internet servers and two households are connected by the layer 3 router `r1`, implemented in P4. Due

[1] https://github.com/ralfkundel/p4-codel

to the bottleneck link between r1 an s2 a CoDel based queueing in r1 with a rate of the bottleneck link speed is installed. The bmv2 only supports a rate limit in packets per second, therefore we provide the measured values in pps with a fixed packet size of 1514 bytes. This is a limitation caused by the bmv2 reference model and a “bytes-per-second” implementation could be realized without any changes of the P4 code. For all shown results the dequeueing rate is 2000 pps, 4 TCP flows are used in parallel and the RTT of the links is 4 ms.

For testing the behavior of the CoDel implementation iperf3 was used between h1 and h3. h2 and h4 provide additional capabilities for parallel ping tests on non-busy hosts.

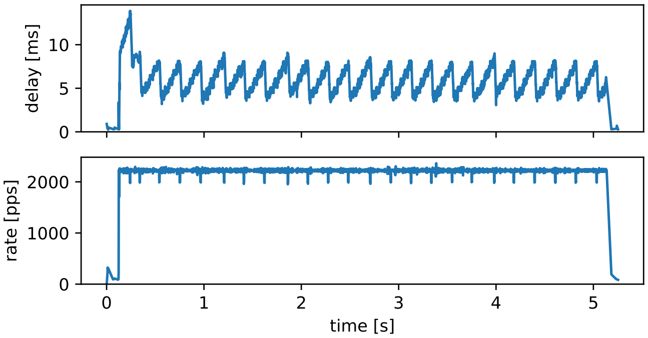

Fig. 5. Ingress to Egress delay and throughput of r1

The correctness of our implementation is shown by the following CoDel specific properties: (1) the queue delay is falling below the TARGET of $5ms$ periodically as shown in Figure 6, (2) at the beginning a burst of packets is caught and (3) the throughput of the TCP connection is constantly equal to the speed of the limiting link (Figure 5). As shown in the results, TCP can not bloat the buffer as known from typical taildrop or RED queues. The results can be reproduced on any linux system. Please follow the instructions in the readme file of the GitHub repository.

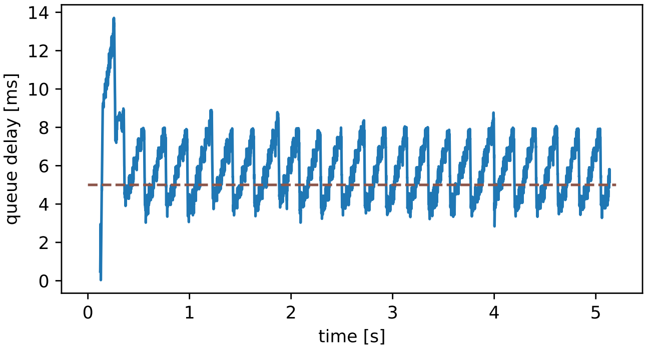

Fig. 6. Measured queueing delay within the P4 pipeline of r1

V. Conclusion

Active queue management algorithms are very promising and powerful to cope with the bufferbloat problem. In this paper we have demonstrated, on the example of CoDel, that it is possible to implement such algorithms for P4 programmable data planes without the need of special hardware. Our open-source available implementation has shown the expected CoDel behavior and can be used for further investigation, evaluation and research. We are firmly convinced that active queue management in the data plane will contribute to reduce latencies in core and edge networks and thereby increase the performance and quality of experience in the future Internet.

Acknowledgment

This work has been supported by Deutsche Telekom through the Dynamic Networks 7 project, and in parts by the German Research Foundation (DFG) as part of the project C2 within the Collaborative Research Center (CRC) 1053 MAKI. Furthermore, we thank our colleagues for their valuable input and feedback.

References

[1] C. Labovitz, D. McPherson, S. Iekel-Johnson, and M. Hollyman, “Internet traffic trends,” in *NANOG*, vol. 43, 2008, p. 2008.

[2] M. Allman, V. Paxson, and E. Blanton, “Tcp congestion control,” RFC5681, https://www.rfc-editor.org/rfc/rfc5681.txt, RFC, 2009.

[3] R. Bush and D. Meyer, “Some Internet Architectural Guidelines and Philosophy,” Internet Engineering Task Force, Request for Comments 3439, 2002.

[4] G. Appenzeller, I. Keslassy, and N. McKeown, *Sizing router buffers*. ACM, 2004, vol. 34, no. 4.

[5] “Global ping statistics,” https://wondernetwork.com/pings, accessed: 2018-07-10.

[6] J. Gettys and K. Nichols, “Bufferbloat: dark buffers in the internet,” *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.

[7] K. Nichols, V. Jacobson, A. McGregor, and A. Iyengar, “Controlled Delay Active Queue Management,” Internet Engineering Task Force, Request for Comments 8289, 2018.

[8] R. Pan, P. Natarajan, F. Baker, and G. White, “Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem,” Internet Engineering Task Force, Request for Comments 8033, 2017.

[9] F. Schwarzkopf, S. Veith, and M. Menth, “Performance analysis of codel and pie for saturated tcp sources,” in *Teletraffic Congress (ITC 28), 2016 28th International*, vol. 1. IEEE, 2016, pp. 175–183.

[10] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, “Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn,” in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 99–110.

[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[12] B. Vinnakota, “Netronome and p4: A brief history and a roadmap,” https://www.netronome.com/blog/netronome-and-p4-a-brief-history-and-a-roadmap/, accessed: 2018-07-06.

[13] “Barefoot networks tofino,” https://barefootnetworks.com/products/brief-tofino/.

[14] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, “Approximating fair queueing on reconfigurable switches,” in *USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[15] C. Cascone, N. Bonelli, L. Bianchi, A. Capone, and B. Sansò, “Towards approximate fair bandwidth sharing via dynamic priority queuing,” in *Local and Metropolitan Area Networks (LANMAN), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–6.

[16] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan, “No silver bullet: extending sdn to the data plane,” in *Proceedings of the Twelfth ACM Workshop on Hot Topics in networks*. ACM, 2013, p. 19.

[17] K. Kaur, J. Singh, and N. S. Ghumman, “Mininet as software defined networking testing platform,” in *International Conference on Communication, Computing & Systems (ICCCS)*, 2014, pp. 139–42.