The Virtue of Gentleness: Improving Connection Response Times with SYN Priority Active Queue Management

Tristan Braud, Martin Heusse, Andrzej Duda

The Virtue of Gentleness: Improving Connection Response Times with SYN Priority Active Queue Management

Tristan Braud*, Martin Heusse¶, and Andrzej Duda¶

{braudt@ust.hk, martin.heusse@imag.fr, andrzej.duda@imag.fr}

*Department of Computer Science and Engineering, Hong Kong University of Science and Technology.
¶Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France.

***Abstract*—We have analyzed network traces of TCP connections and observed that there are many more losses during the handshake than for the remainder of the data exchange. Although recently developed AQM schemes can efficiently reduce latency related to *bufferbloat*, only more complex solutions relying on Fair Queueing (FQ) can improve the long delays resulting from the loss of a packet during the establishment of a TCP connection. In this paper, we propose SPA (SYN Priority Active queue management), a new low-complexity queue management scheme that combines the benefits and simplicity of the most recent AQM schemes while achieving performance comparable to more complex combinations of Fair Queueing and AQM. Our evaluation shows that the SPA performance is close to FQ CoDel for only a fraction of the complexity and resource usage.**

I. Introduction

Shortening the response time and reducing overall latency of TCP transfers is paramount to improve the responsiveness of information access over the network. In this paper, we focus on the connection establishment phase, which is often the limiting factor of the transfer response time, because at the beginning, connections do not have an estimate of the round trip time (RTT) between the client and the server, so they use a long default retransmission timeout (RTO). The reception of SYN/ACK immediately updates RTO so that all other segments can be quickly resent, even without Fast Retransmit. Thus, the client can only recover from the loss of the initial SYN segment or the corresponding SYN/ACK after a long period several orders of magnitude larger than the recovery time of subsequent data segments. This effect is particularly detrimental to short connections. For larger ones, the impact of a SYN loss becomes less significant, especially in the case of non-interactive traffic. Despite the importance of the connection establishment phase, speeding it up has received less attention than ensuring the good performance of the bulk data exchange [1]. Over the past years, many proposals aimed at maximizing link utilization and achieving low delays, the most well-known being *CoDel* [2] and *FQ CoDel*, its Fair Queueing counterpart [3]. However, the most recent packet scheduling schemes [2], [4] present the drawback of either increasing SYN drops, or relying on Fair Queueing, which results in higher CPU and memory usage.

To evaluate the impact of SYN or SYN/ACK losses on the transfer response time, we have analyzed a set of publicly available real world traffic traces. The analysis shows that SYN and SYN/ACKs are generally lost much more often than other TCP segments, which is another reason to consider them in a particular way. We would expect that the SYN retransmission rate is twice that of regular data segments, because it happens after a SYN loss and also after a SYN/ACK loss. In fact, we observe an even much higher SYN retransmission rate: several times the retransmission rate of all segments, which exacerbates the performance problem for short connections. SYN/ACK losses has a detrimental impact on performance: their retransmissions cause a two-fold increase of the retransmission delays experienced by connections. Based on these observations, we propose SPA (SYN Priority Active queue management), a queueing scheme with two packet queues both managed by *CoDel* (Controlled Delay Management) [2]: a higher priority queue handles SYN and FIN packets, and a lower priority one manages data segments. SPA is thus a pair of queues with independent AQM mechanisms. It is much less complex than any fair queueing scheme and results in short connection setup times and smooth low latency data transmission. Similarly to CoDel and FQ CoDel, this scheme is designed to be deployed at the "last mile" of the network, typically in home routers, where resources (CPU and bandwidth) tend to be scarce, and reactivity is key.

The contribution of this work is threefold: first, we analyze network traces to get insight into the behavior of the TCP establishment phase and observe a significantly higher SYN retransmission rate compared to other types of segments. Second, we model the effect of SYN retransmissions on transfers of a limited size to show that SYN losses account for a large part of the response time. Finally, we evaluate SPA through measurements on a testbed and compare its performance with recent and well-established scheduling mechanisms. We show that under normal traffic conditions, SPA performs similarly to FQ CoDel, for a fraction of complexity. We also shed light on a case for which FQ Codel collapses due to its own design.

II. Related Work

The delay introduced by the connection establishment phase is often overlooked in the literature even in recent papers [5], [6]. Some authors observed that the loss probability of SYN segments and regular ones may differ [7], although without exploring the impact of this difference. Ciullo et al. [8] observed the distribution of the connection completion time and realized that more than 70% of dropped packets

ISBN 978-3-903176-08-9

are recovered after RTO. They also proposed two schemes to overcome long recovery delays due to the loss of the last data segment in a flow. Anelli et al. [1] made a case for protecting SYN segments to improve connection response times. They introduced *REDFavor*, a RED mechanism with a specific processing of SYN segments. Another idea proposed in the literature is to resend aggressively SYNs to avoid the impact of the long timeout on lost SYN segments [9]–[11]. Although functional, this last solution requires the user to modify the operating system or use additional software, while adding some traffic in an already congested link. Similarly, solutions such as WonderShaper [12] or DD-WRT [13] also take the approach of giving a higher priority to all control messages. However, they rely on FIFO queues that requires prior configuration to achieve low latency. This approach cannot be efficiently used in delay or bandwidth-varying environments.

Recently, researchers have started to discuss the *bufferbloat* effect, the problem of long delays due to excessive buffer sizes in access networks [14]. A workgroup on bufferbloat began in 2011 [15] and some analyses of the problem started to appear [16]–[19]. Two main proposals for solving the problem include *CoDel* [20] and *PIE* (Proportional Integral controller Enhanced) [4], [21]. CoDel measures the packet sojourn time in the queue and drop packets at the queue head to keep the delay from exceeding a reasonable value for any significant period of time. It requires per packet timestamps, but it does not need any configuration parameter except for the default *threshold delay*. *FQ (Fair Queueing) CoDel* [3] extends CoDel with the principles of *Stochastic Fair Queueing* [22] to manage per flow queues and mitigate the adverse effects of purely random packet drop. PIE [21] controls the average queueing latency so it stays at a reference value. It combines the benefits of both RED and CoDel: PIE randomly drops a packet at the beginning of the congestion detected based on the queueing latency like CoDel. PIE can ensure low latency and achieve high link utilization under various congestion conditions. Schwardmann et al. evaluated by simulation the robustness of CoDel, PIE, and ARED for various static and dynamic scenarios [23] in a simple set-up with one link and a variable number of bulk TCP flows. *ARED* (Adaptive RED) is an active queue management scheme that attempts to stabilize the average queue size around some preset target queue size [24]. PIE achieved lower delays than CoDel and ARED for low capacity links, but for higher capacity links, CoDel and ARED resulted in lower delays. In dynamic scenarios, CoDel results in a lower maximum delay than the other schemes. Khademi et al. evaluated CoDel, PIE, and ARED [25], but in an experimental setup using Linux implementations in a wired testbed for bulk TCP transfers. The authors also observed that the CoDel "dropping-mode" interval needs to be set lower than the default value, while we note in this paper (see Section VI-B), the noticeable influence of the target delay that it uses in the case of the FQ CoDel variant. These results contradict the claim that CoDel is a parameterless AQM. Otherwise, ARED performed the best in the Khademi et al. study except when the number of flows on the bottleneck link was very small.

III. Analysis of Real World Traces

To evaluate the impact of SYN or SYN/ACK losses, we have analyzed a set of publicly available traffic traces: five

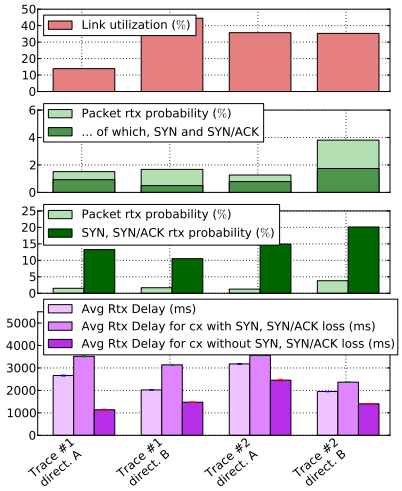

Fig. 1. Analysis of the CAIDA traces in both directions ordered by ascending order of link utilization in direction A. SYN losses represent a large part of all losses and SYN segments have a significantly higher loss probability than data segments leading to a significant impact on retransmission delays.

sets of traces from the CAIDA data set[1] recorded on two different days at an Equinix datacenter in Chicago connected to a 10 Gigabit Ethernet backbone link of a Tier 1 ISP. As the traces from CAIDA are split according to the direction, we analyze them in one direction at a time. We consider two days with differing traffic load patterns: for the trace taken on March 20th, 2014 the link utilization in direction A is markedly lower than on September 18th, 2014. For readability purposes, we will further refer to the traces as Trace "Real" and Trace "Short" respectively. All measurements except the link utilization concern connections with at least one SYN, which represents from 180,000 to 650,000 connections per trace. We have filtered out the connections with retransmission delays greater than 45s to remove inconsistent data from the analysis (the value of 45s corresponds to the total delay after loosing 3 SYNs at the start of a connection due to the exponential backoff[2]). Due to the number of samples, the 95% confidence interval could not be represented for loss probabilities and is barely visible for retransmission times.

Figure 1 shows two striking phenomena:

1) SYN and SYN/ACK retransmissions account for a large part of the retransmissions (2^{nd} plot)
2) the probability of SYN or SYN/ACK retransmissions is exceptionally high—between 10% and 20% (3rd plot). The connections with SYN or SYN/ACK retransmission suffer from an average retransmission delay almost twice that of other connections (4th plot).

Note that the initial RTO is set to 3 seconds by default on Windows, FreeBSD, and Linux (prior to 2011; it is now 1 s). 10% to 20% of all connections are affected whereas the packet loss rate remains limited at 1% to 3.5%.

[1]The CAIDA UCSD Anonymized Internet Traces 2014 *20/03/2014 12:59:11, 13:03:00, 13:06:00 and 18/09/2014 13:07:00, 13:19:00* http://www.caida.org/data/passive/passive_2014_dataset.xml

[2]as stated in the FreeBSD source code *"the odds are that the user has given up attempting to connect by then."* [26].

TABLE I. ANALYSIS OF MAWI TRACES

	Dir A	Dir B
Packet rtx prob.	1.65%	2.73%
...of which, SYN, SYN/ACK represent	29.8%	79.5%
SYN and SYN/ACK rtx prob.	13.1%	29.9%
Average rtx delay (ms)	5276.96	2114.93
Average rtx delay for connections with SYN or SYN/ACK loss (ms)	6324.93	2390.63
Average rtx delay for connections without SYN or SYN/ACK loss (ms)	4707.11	1011.93

TABLE II. MAWI TRACES, SYN FLOOD FILTERED OUT

	Dir A	Dir B
Packet rtx prob	1.63%	2.72%
...of which, SYN, SYN/ACK represent	28.8%	78.9%
SYN and SYN/ACK rtx prob	12.8%	40.6%
Average rtx delay (ms)	5312.9	2114.93
Average rtx delay for cx with SYN or SYN/ACK loss (ms)	6469.4	2390.63
Average rtx delay for cx without SYN or SYN/ACK loss (ms)	4710.5	1011.93

The higher retransmission rate of SYNs compared to other segments is expected as a SYN/ACK drop in the reverse direction always causes a timeout, whereas cumulative ACKs make the established connections relatively immune to losses on the return path [27]. So, as long as SYN and SYN/ACK segments are as likely as other packets to be dropped, the retransmission rate doubles. Since the queues in many routers are managed as *packet FIFOs*, small SYN packets are just as likely to be dropped as the larger ones. On the contrary, a *byte-based FIFO* would favor small packets that can generally still fit in the queue when larger packets are dropped. In our trace analyses, we do not see any situation in which small packets would be retransmitted less than larger ones, which means that most routers use packet FIFOs. (If some routers use byte FIFO, it is beneficial to small flows as shown below.) However, the fact that retransmissions occur for losses in both directions only explains a small fraction of the observed difference. Another explanation of the discrepancy between SYN and regular segment losses could be TCP congestion control algorithms. They are designed so that connections loose a limited number of segments per congestion episode (1 for the congestion avoidance phase). After a loss, TCP divides the congestion window by 2 to reduce the transmission rate and delay the next occurrence of congestion. Subsequent packets are thus less likely to experience another loss. In contrast, SYN packets arrive at random and their potential retransmission is so distant in time that the conditions are uncorrelated, which could lead to more losses than regular packets. Again, more investigations are required to explain this difference.

To confirm the findings, we have also analyzed traces collected by the MAWI Working Group of the Wide project on the sample point F (1Gb/s transit link between WIDE and an upstream ISP). Again, we filter out connections with retransmission delays greater than 45s (see Table I). We can first observe that one direction presents less losses than the other one although, according to the retransmission delays, connections experience many timeouts. We have observed the same phenomenon in the CAIDA traces taken on September 18, 2014 in direction A: although losses are less frequent than in direction B, most of them end up as a timeout. The reason may be bufferbloat in an upstream buffer that causes higher delays, thus setting off the RTO timer. The observed delays corroborate the results from the CAIDA traces: SYN and SYN/ACK losses account for almost half of the losses. SYN and SYN/ACK retransmissions cause a two-fold increase of the retransmission delays experienced by connections.

As we analyze connection establishment, one concern is that the presence of SYN flood attacks may bias the results. We have filtered out potential SYN floods by removing

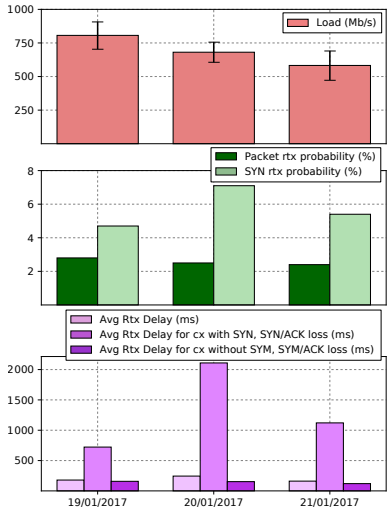

Fig. 2. Analysis of the bidirectional MAWI traces ordered by date for connections with a full three way handshake and a data packet. Although SYN losses represent a lower part of total losses, they still have a higher loss probability than data segments leading to a significant impact on retransmission delays.

connections involving IP addresses that show the signs of a SYN flood, which is possible for bidirectional MAWI traces. To identify them, we have first counted the connections that effectively carry data segments for each IP address present in the trace. We have filtered hosts that showed strong signs of being under a SYN flood attack (or could have been dysfunctional), when they experienced 1000 times more connections without data than with data (with a minimum of 1000 connections without data). We have removed 69 out of 587,262 IP addresses present in the trace, which results in removing over 30% of connections mainly in direction B. Table II presents the corrected data. In direction A, we have removed only a few connections and the results are mostly unchanged. Removing a large part of connections in the reverse direction dramatically increases the SYN retransmission probability up to 40%, while the delays and the retransmission probabilities are not significantly affected. Indeed, in the case of a SYN flood, we expect to see a large amount of unique SYNs, without any further segment or retransmission, as the attacker does not have any interest in establishing a real connection. As the MAWI traces are bidirectional, we can now focus on the analysis of connections that manage to pass the three-way handshake and transmitted at least one data segment. Figure 2 presents the results. Fully established connections present more contrasted results. Indeed, the SYN loss probability is now only twice as big as the packet loss probability, which

corroborates the hypothesis we have made earlier, stating that we should observe a SYN loss probability twice the packet loss probability as the loss of a SYN/ACK packet will trigger a SYN retransmission from the other side. Although we observe less SYN losses, their impact on the average retransmission time is much higher: between 10% and 40% of the average retransmission time is due to connections with at least a SYN loss. Such connections also present average retransmission times 4 to 14 times higher than connections without a SYN loss. Our findings from both data sets are in line with the results by Damjanovic et al. in the LBNL/ICSI Enterprise Tracing Project [9] that showed an overall 10% of SYN retransmissions for all connections in a LAN and 2% of SYN retransmissions for successful connections. Still, we observe much higher loss rates (10–20% overall SYN loss and 5–8% for successful connections). This difference could be explained by the fact that the CAIDA and MAWI traces come from a transit link where it is more likely to include 3/4G, WLANs, or even satellite traffic with variable delays and random losses, or simply more congestion along the way. Such conditions are more prone to generate losses, create timeouts, and in general, create more fluctuations in the analyzed values.

IV. Model of the TCP Response Time under SYN Losses

To evaluate how SYN losses impact short TCP transfers, we propose to estimate the transfer duration based on the well-known model proposed by Mathis et al. [28] and Padhye et al. [29], extended to take into account SYN retransmissions.

We first consider the congestion avoidance phase of a TCP connection with constant *RTT* and subject to packet loss probability p_l. The congestion window oscillations are equivalent to cyclic oscillations between $\frac{\bar{W}_{\max}}{2}$ and $\bar{W}_{\max}$ so the average expected window is:

$$\bar{W}_{\text{avg}} = 3/4 \times \bar{W}_{\max}$$

and the expected throughput in pkt/s is the following:

$$\bar{X} = \frac{3}{4} \times \frac{\bar{W}_{\max}}{RTT} \approx \frac{1}{\text{RTT}} \sqrt{\frac{3}{2\,p_l}},$$

using Mathis' estimate for $\bar{W}_{\max}$. We model the slow start phase by considering an initial congestion window of one segment going up to $\bar{W}_{\max}$. We approximate the number n_{ss} of RTT taken by the slow start phase as $2^{n_{\text{ss}}-1} = \bar{W}_{\max}$, so that

$$n_{\text{ss}} = \frac{log(\bar{W}_{\max})}{log(2)} + 1.$$

Now, we need to estimate the time spent in recovery of losses that may happen during the connection. For one loss per congestion event that happens with probability p_l and recovery time R_t, a connection of size S MSS spends time $t_{\text{recovery}} = S \times R_t \times p_l$ in recovery, knowing that optimistically, $R_t \approx RTT$. If the probability of a SYN loss is p_{Sl}, the time spent in the connection establishment phase is:

$$t_{\text{syn}} = RTT + RTO_0 \sum_{k=1}^{\infty} (p_{\text{Sl}})^k = RTT + \left(\frac{1}{1-p_{\text{Sl}}} - 1 \right) RTO_0, \quad (1)$$

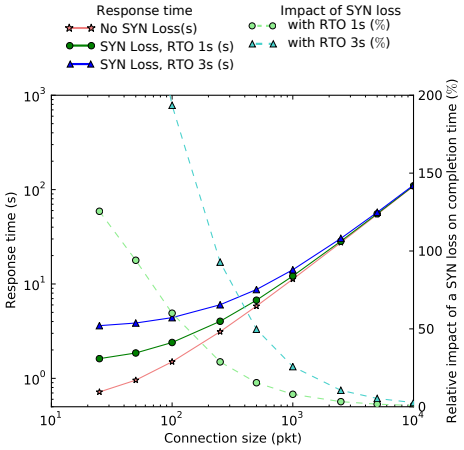

Fig. 3. Response time of connections with and without SYN loss

where RTO_0 is initial retransmission timeout. We can approximate p_{Sl} with $\approx 2p_l$ or obtain it from measurements.

For a connection of size S, the total response time is:

$$t_{tot} = t_{\text{syn}} + t_{\text{slow\_start}} + t_{\text{congestion avoidance}} + t_{\text{recovery}} + t_{\text{fin}}, \quad (2)$$

where $t_{\text{slow\_start}} = n_{\text{ss}} \times RTT$, $t_{\text{congestion\_avoidance}} = \frac{S-(2^{n_{\text{ss}}}-1)}{\bar{X}}$ and $t_{\text{fin}} = RTT$.

Figure 3 represents the estimated response time of connections with and without SYN loss with RTO of 1 s or 3 s, for the connection size between 25 and 10000 segments in the conditions of Trace "Real" (Figure 1). A SYN loss is a major problem for short connections as it increases the response time by up to 200%. The impact becomes negligible (i.e., $< 10\%$ of the response time) for connections larger than 1000 segments for RTO of 1 s and 2000 segments for RTO of 3 s. Short connections, which are generally already penalized compared to long connections, dramatically suffer from this phenomenon.

V. SPA – SYN Priority AQM Scheme

CoDel [2], an almost parameterless, delay-based queueing management scheme, is a recent solution to bufferbloat. However, as other AQMs designed to tackle excessive queueing delays, CoDel assumes that all losses are good to the network, as long as they contribute to keeping the delay low. Yet, many losses will always result in timeouts, especially head drops—when the initial TCP RTO is still up to make matters worse—and tail drops can also result in long timeouts if the connection presents some delay fluctuations. FQ CoDel [3] solves this problem by giving a higher priority to the first packets, but at the cost of a higher complexity and memory footprint: for instance, the default Linux implementation uses 1024 separate queues. Our approach is to strike a compromise: we manage only two packet queues by CoDel—a higher priority queue for SYN and FIN packets and a lower priority one for regular packets, and do not keep track of connection states. The pseudocode below defines the scheme more formally:

SPA (SYN Priority AQM) Scheme

```
synfinqueue = CoDel()
packetqueue = CoDel()
def enqueue(p):
```

```
    if p.flag.SYN==set or p.flag.FIN==set:
      synfinqueue.enqueue(p)
    else:
      packetqueue.enqueue(p)
  def dequeue(p):
    if synfinqueue.is_empty():
      packetqueue.dequeue(p)
    else:
      synfinqueue.dequeue(p)
```

Three types of losses may lead to a timeout: SYN, FIN, and tail losses. While it is difficult to detect tail losses at the router level, it is much easier to isolate SYN and FIN in a separate queue and avoid timeouts. The queue for SYN and FIN has priority over the second one for data packets and both queues are managed by CoDel. In this way, we significantly increase the overall performance by preventing most of timeouts, while keeping complexity much lower than FQ CoDel. We can easily implement SPA based on an existing CoDel implementation in coordination with regular system tools. For our experiments, we have set up SPA with a priority queue serving two CoDel subqueues using `tc` [30] and `iptables` for packet filtering. As mentioned before, another possibility to avoid SYN and SYN/ACK losses is to dimension the buffer in bytes rather than in packets. Although effective and relatively straightforward to implement, this solution still presents one major drawback: it requires to know in advance the link capacity, whereas in many cases, it is unknown and fast varying, as for WiFi or cellular networks. Excessive queueing space—bufferbloat—results in high latency in the network or conversely, an insufficient buffer may result in low link utilization. By design, the proposed SPA scheme does not suffer from such limitations.

As the SPA scheme favors SYN packets over data packets, it may raise some security concerns. We do not want an attacker exploit our solution to favor her/his connections or to accelerate SYN DoS attacks. Regarding the first issue, a basic attack would be to set the SYN or FIN flag on every data packet. Admitting that the receiver does not discard such packets, checking the packet length is enough to prevent abuse. SYN DoS attacks raise a more complex issue. Indeed, as SPA favors SYN packets, a router could be used as a relay to accelerate the transit of SYN packets at the bottleneck and amplify a SYN flood attack. However, similarly to CoDel and FQ CoDel, SPA is intended to be deployed at the bottleneck, usually at the last mile of the network, where an attacker may already have full control of the link. For instance, deploying SPA in home routers is harmless. Most Fair Queueing solutions are also vulnerable to this type of exploit, as they prioritize packets from new connections, since they constantly try to equalize the service received by all connections, and the new one initially shows a deficit. A batch of SYN packets with various source addresses and ports would typically be favored over data packets from existing connections. We show this phenomenon in Section VI-B, where FQ CoDel gets swarmed by a large number of new connections, resulting in abnormal behavior, whereas SPA proves to be more resilient.

VI. Testbed Experiments and Performance Comparisons

We have run a series of experiments on a testbed network composed of three computers (cf. Figure 4). One of them acts as a router interconnecting two others, while the two ends are used as a client and a server. This simple topology allows observing the behavior of various queueing policies at the bottleneck. The hosts run FreeBSD 9.3 that allows us to test the latest variants of TCP without switching operating systems. The router runs Debian GNU/Linux (wheezy) with a slightly modified 3.18 kernel as described below. On top of this operating system, we run the `tc` tool to emulate links and various queueing management policies. We have set the kernel context switching frequency to 1kHz to emulate the bottleneck link smoothly for the bit rates up to 10Mb/s. We use the `ipmt` suite [31] to generate connections as follows: connection arrival times follow a Poisson process with parameter λ and connection sizes follow a Zipf distribution with parameter μ. We have run experiments with and without reverse traffic.

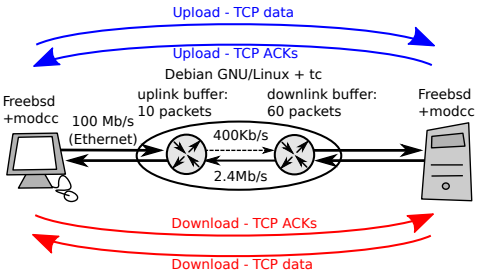

Fig. 4. Testbed network for experiments. The central computer emulates the bottleneck link in both directions and implements the considered policies. The traffic is in the downlink direction with or without a long lasting upload TCP data transfer.

TABLE III. Experiment parameters

Uplink		
Capacity C_u	0.4Mb/s	2Mb/s
Delay D_u	52ms	
Scheduling Q_u	Packet FIFO	
Downlink		
Capacity C_d	2.4Mb/s	10Mb/s
Delay D_d	42ms	
Scheduling Q_d	Packet FIFO - Byte FIFO RED - ARED - REDFavor CoDel - FQ CoDel – PIE SFQ SYN Prio - SYN/FIN Prio - SPA	
Connection Generation Parameters		
Traffic Type	**"Real"**	**"Short"**	**"10M"**
$1/\lambda$ (s)	0.12	0.017	0.032
μ	1.7	2	1.7

We run experiments under three different traffic conditions shown in Table III. As SPA targets the last mile section of a network, we consider the case of typical ADSL link as it is still far more deployed than fibers [32]. We use a 42 ms round trip delay, which corresponds to the ADSL interleaving delay (typically 24 ms) plus an intra-continental propagation time [33]. The slightly higher uplink delay helps to avoid perfect synchronization between the upload and download feedback loops. The traffic conditions correspond to a high link utilization of 90%. By varying the λ and μ parameters as well as the connection size, we obtain various traffic conditions:

Traffic Type "Real" corresponds to realistic assumptions with connection sizes distributed according to a heavy tailed Zipf distribution: $1/\lambda = 0.12s$, $\mu = 1.7$, which leads to an average

size of 21 segments per connection, a median of 2 packets and 80% of connections spanning 6 segments or less. This load is applied with and without reverse traffic.

Traffic Type "Short" represents extreme conditions to push the tested schemes to the limits. We generate a connection every $1/\lambda = 0.017s$, connections have the average size of 3 segments ($\mu = 2$), a median size of 1 segment, and the maximum size of 100 segments (80% are 4 segments or less)

Traffic Type "10M". With this type, we analyze the behavior of the tested schemes for a link with the increased capacity to 10Mb/s with 95% load composed of connections arriving on the average every 0.032 second, with an average size of 26 segments and a median of 2 segments (80% under 7 segments).

We test the following scheduling schemes at downlink queue Q_d: regular FIFO (limited either in packets – Packet FIFO or in bytes – Byte FIFO), AQM (RED, ARED [24]), delay controlled mechanisms (CoDel and PIE [4]), Fair Queueing (FQ CoDel and SFQ – Stochastic Fair Queueing [22]). We compare these queueing schemes to priority queues with a higher priority given to SYN or SYN/FIN (SYN Prio, SYN/FIN Prio, RedFavor [1], and SPA). The uplink queue remains unchanged for all experiments (Packet FIFO). We use the default threshold delay of 5 ms for CoDel, FQ CoDel, and SPA. As we have discovered that in some cases, this parameter needs to be tweaked for CoDel and FQ CoDel, we also try the value of 20 ms. We did not include solutions at the edge like setting the SYN retransmission timeout to a more aggressive value [9]–[11], because they may only improve the retransmission time, but they cannot lower the SYN loss rate.

A. *Results for Traffic Type "Real"*

Figure 5 presents the results for Traffic Type "Real". Similarly to Figure 1, due to the number of connections (8326), we are only able to display 95% confidence intervals for the retransmission and response times. The two most prominent outcomes are: (1) all schemes provide better response times than Packet FIFO and (2) only a few schemes clearly stand out: FQ CoDel, SFQ, RedFavor, and SPA. **For short connections, FQ CoDel 20ms and SPA are the best ones**. SFQ attains the best result for longer connections with REDFavor and SPA also having short response times. We observe remarkable differences of RTT between delay-based solutions and regular queueing schemes: CoDel and PIE halve the RTT compared to FIFO or RED, while FQ CoDel and SPA achieve the lowest RTTs. We also see that about one fourth of the average retransmission delay is contributed by connection SYN losses: looking at the traces, a small fraction of connections experience a 3 second delay before even being able to send a single packet. Byte FIFO gets shorter retransmission delays (20%) than Packet FIFO, reaching the same performance as RED with almost no SYN loss and a lower RTT than Packet FIFO. **Byte FIFO even outperforms RED, ARED, CoDel, and PIE in terms of response times!** SFQ shows unusually high retransmission delays. Indeed, even if there are almost no SYN losses, most losses result in one and often multiple timeouts. In essence, the **Fair Queueing algorithm leads to the starvation of long flows**: as the queue experiences high load with many short connections, the longer ones are not serviced and eventually timeout. Yet, Fair Queueing policies

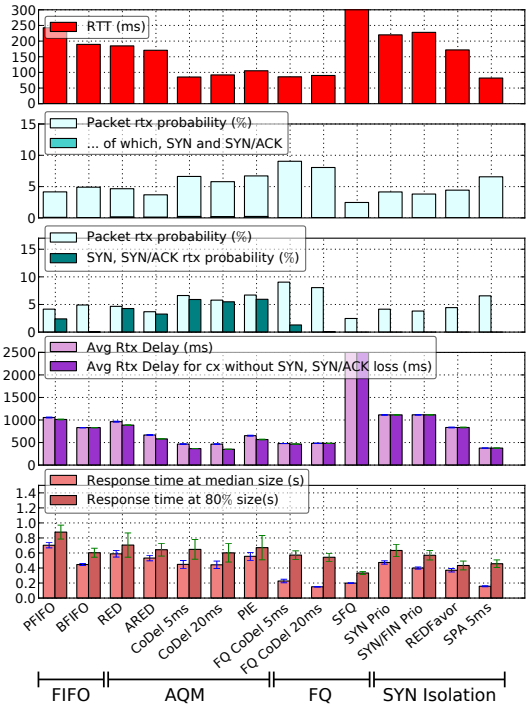

Fig. 5. Average RTT, loss percentage, retransmission delays and response times for various queueing schemes under Traffic Type "Real". FQ CoDel, SFQ, RedFavor, and SPA achieve the shortest response time.

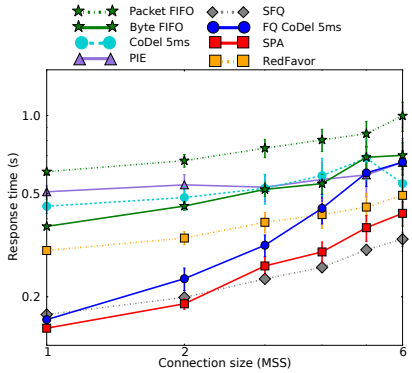

Fig. 6. Response times of connections for various queueing schemes under Traffic Type "Real". SPA and Fair Queueing schemes result in significantly lower response times compared to other AQM and FIFO.

display some of the best response times for short connections. Considering the isolation of SYN and/or FIN in a separate FIFO (SYN and SYN/FIN Prio, REDFavor), we do not observe a real improvement in RTT and retransmission times compared to basic FIFO. Nevertheless, there are fewer SYN retransmission, which improves the response times especially for short connections. Finally, **SPA results in the shortest retransmission delay**, while keeping one of the lowest average RTT and response times. FQ CoDel achieves a similar though slightly worse performance if left with default parameters.

Figure 6 presents the connection response times for a selection of the considered policies. There is a manifest separation between two groups: in the first one, PIE and CoDel exhibit similar performance, while **Byte FIFO shows a noticeable improvement compared to Packet FIFO** (connections complete

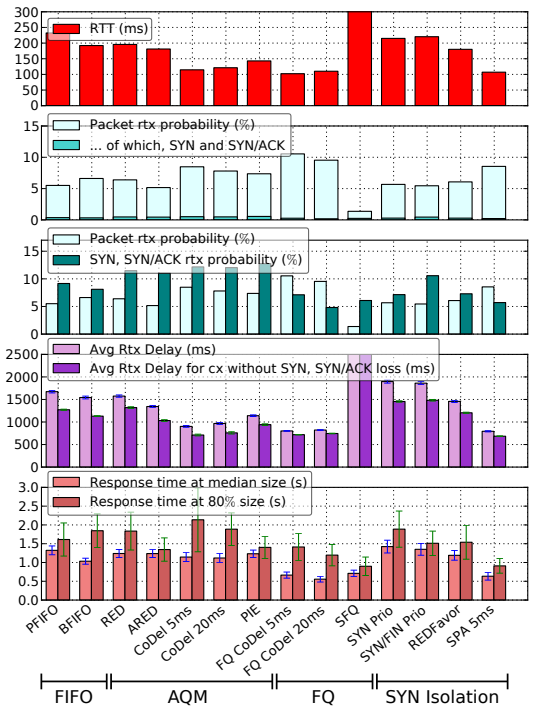

Fig. 7. Average RTT, loss percentage, retransmission delays, and response times under Traffic Type "Real" with one long reverse connection. SYN retransmissions rate, retransmission delays, and response times raise drastically. FQ CoDel and SPA achieve the shortest response times for short connections, while SFQ and SPA are the best for longer connections.

40% faster). In the second group, **SPA, SFQ, and FQ CoDel present lower response times** with an improvement between 100% and 300% for the shortest connections. In the case of delay based solutions like CoDel or PIE, this plot confirms that the philosophy of "*loss is good*" is detrimental when control segments such as SYN are not considered in a specific way: the schemes lead to high data segment retransmission rates, which is actually good for reducing congestion and delays, but also to high levels of SYN retransmissions. Consequently, the schemes experience longer retransmission delays and longer delays, even if they still manage to improve the performance over Packet FIFO. REDFavor performs better than other AQM schemes: with this simple modification, response times are halved compared to Packet FIFO, which is a quite noticeable enhancement compared to other AQM mechanisms. Finally, our solution performs better than FQ CoDel and SFQ for short connections. For connections with more than 3 segments, the difference between the regular and modified FQ CoDel becomes less significant and connections under SPA complete in 30% less time than with FQ CoDel. SFQ continues to obtain good response times similar to those obtained with SPA, but with inacceptable retransmission times for larger connections, as we can see in Figure 5.

We complement the first experiment that dealt with unidirectional traffic by adding one long reverse connection. The idea is to approach "near real" traffic conditions as encountered for instance behind an ADSL link with a user uploading files to a Cloud service provider. The results appear in Figures 7 and 8. The first expected result is an increase of SYN losses in the presence of reverse traffic. Indeed, SYN/ACKs can be

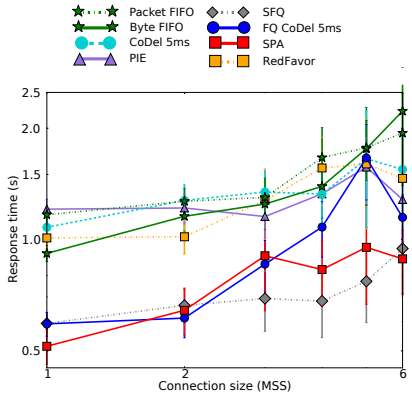

Fig. 8. Response times of connections for various queueing schemes under Traffic Type "Real" with one long reverse connection. Important gap between AQM and Fair Queueing schemes, but more differences inside each group.

lost on the return path, while the presence of ACKs in the download queue virtually reduces the queue size and leads to an overall increase of losses [34]. We observe 25% more losses on average and twice as many SYN losses. Consequently, SYN retransmissions account for a larger part of the retransmission delay: connections without a SYN loss experience retransmission delays 25% lower than average. Similarly, response times are much longer. We see that for short connections, FQ CoDel and SPA achieve the shortest response times, while SFQ and SPA are the best for the longer connections. SPA has once again similar performance to FQ CoDel, both in terms of RTT and retransmissions times, with response times halved compared to most of the AQM mechanisms.

Concerning response times (cf. Figure 8), we observe the same phenomenon: even if connections take much more time to complete compared to the situation without reverse traffic, we can still differentiate between two groups: FQ CoDel, SFQ, and SPA obtain much better results than other queueing schemes. In the AQM group, the differences are small except for SPA that performs similarly to the other Fair Queueing schemes. The values for larger connections suffer from the bias introduced by heavy tail distributions: many very long connections do not terminate during the measurement session and those that finish benefit from shorter response times. This bias explains lower response times for connections of 6 MSS.

B. Traffic Type "Short"

In the next experiment involving Traffic Type "Short", extreme conditions (majority of very small connections) push the tested schemes to the limits with more frequent SYN losses. Figure 9 shows that **FQ CoDel, SPA, SFQ, and SYN/FIN Priority achieve the shortest response times for short connections**, while SFQ and SYN/FIN Priority are the best for the longer connections. RTT is significantly lower than in the previous experiment. This is due to the size of the transfers: with an average of 3 segments and an initial cwnd of 4 packets, most of the connections only last for the first burst of the slow start phase. Moreover, we have a proportion of 3 small segments (SYN, ACK, and FIN) for 3 data segments in the queue, virtually decreasing the queue size. Byte FIFO achieves shorter response times than AQM schemes, due to the fact that it causes almost no SYN retransmissions. The explanation of

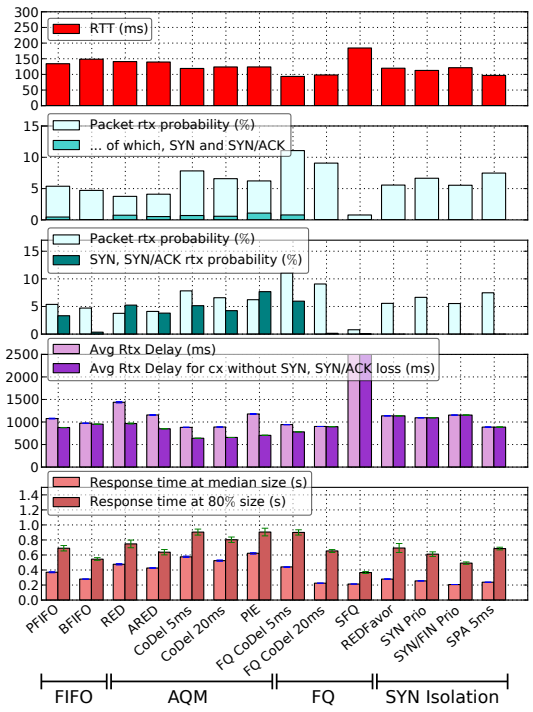

Fig. 9. Average RTT, loss percentage, retransmission, and response times under Traffic Type "Short": heavy tailed distribution with a majority of very small connections. More SYN losses than with Traffic Type "Real": larger delays (RTT and retransmission) for CoDel and PIE. FQ CoDel, SPA, SFQ, and SYN/FIN Priority display the shortest response times for short connections, SFQ and SYN/FIN Priority are the best for longer connections.

more SYN retransmissions for RED queues is inherent to their implementation in Linux: the algorithm computes the average queue length at large time intervals. When congestion appears, established connections stop sending packets, emptying the queue. Due to this interval, the algorithm takes some time to detect that the queue is again below the loss threshold and incoming packets (mostly SYN) still suffer from drops. This effect is specific to the Linux implementation of RED. In general, AQM mechanisms cause many SYN retransmissions (up to 8% of connections for PIE) that directly impact response times. The retransmission delays under SFQ stand out again due to starvation. We also note that **FQ CoDel with the default 5ms threshold delay does not perform well**. In this critical scenario, SYN packets arrive faster than the link speed. Some SYN packets stay longer in the queue than the FQ CoDel base delay and get dropped. Setting the base delay of FQ CoDel to 20 ms gives back the expected performance. SYN isolation techniques significantly improve response times compared to both their single queue counterpart and other AQM schemes. SPA also exhibits a noticeable improvement over CoDel, and performs better than a correctly configured FQ CoDel, with low RTT and retransmission delays, and some of the shortest retransmission times.

In this experiment, the results greatly differ from those for Traffic Type "Real". First of all, **delay based AQM and RED are clearly not designed for this kind of load**, with a high level of losses, and generally higher response times than with basic Packet FIFO. Byte FIFO continues to show an almost negligible amount of SYN retransmissions and remains

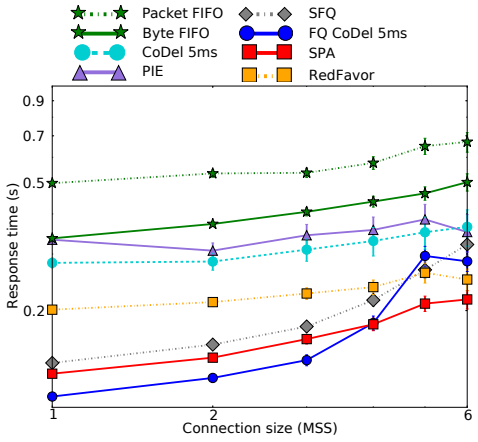

Fig. 10. Response times of connections under Traffic Type "10M" on a 10Mb/s link. FQ CoDel, SPA, and SFQ result in the best performance.

a good alternative to Packet FIFO despite a slightly lower RTT obtained by the latter. **SYN isolation techniques display a significant improvement** in terms of response times, with SYN/FIN Prio and SFQ showing the best performance. Both are closely followed by a correctly configured FQ CoDel and SPA that also obtains lower RTT and retransmission times, thus making them well suited for interactive traffic.

C. Traffic Type "10M"

We have run experiments on a 10 Mb/s link with the parameters similar to Traffic Type "Real" without reverse traffic. Figure 10 presents the results. The experiment is close to Traffic Type "Real" and the results are similar: **FQ CoDel, SPA, and SFQ perform much better than the other schemes**. Even though it is the second worst-performing scheme, Byte FIFO still obtains response times 30% lower than Packet FIFO. The main difference is that PIE and CoDel now improve performance compared to FIFO (whether packet or byte based). Moreover, REDFavor now clearly results in better performance than other AQM schemes, even outperforming SFQ and FQ CoDel for longer connections.

D. Conclusion on the Experimental Results

Based on the four setups above, we first assert that Byte FIFO is generally an interesting alternative to Packet FIFO. However, it is far more complex to implement than many other tested solutions and generally not available. Even though delay based solutions tend to compensate their high levels of SYN losses through lower RTT and faster retransmission times, they usually do not bring much improvement to the general response times and could perform a lot better with a more discerning dropping policy. Finally, isolating SYN and/or FIN in separate queues presents at least similar performance to their single-queue equivalent: in three out of four scenarios, we have observed a considerable improvement with respect to FIFO and RED based solutions. However, **SPA always performs much better than CoDel and similarly to Fair Queueing algorithms** that are drastically more complex. In some cases, it even outperforms those schemes. It does not suffer from starvation, ensuring low RTT and retransmission times in any circumstances. Moreover, it does not break down when facing

an extremely aggressive load (compared to FQ CoDel). It is, by far, the queueing scheme presenting the best performance to complexity tradeoff in our set of experiments.

VII. Conclusion

In this paper, we have evaluated the impact of SYN retransmissions on TCP connection response times based on real world traces. To achieve short response times via protecting SYN and SYN/ACK segments from losses, we have proposed SPA, a new scheme that uses two packet queues managed by CoDel—a higher priority queue for SYN and FIN segments, and a lower priority one for other segments. We have run extensive experiments on a testbed network to compare the most important schemes for three different traffic conditions. Our measurements show that the evaluated AQM mechanisms result in very similar performance: they succeed at maintaining low queueing delays although this result comes at the cost of a significant SYN loss rate. Unsurprisingly, adding some kind of Fair Queueing mechanisms consistently results in good performance, although FQ CoDel may need some tweaking in extreme traffic conditions. SFQ obtains small response times, but most losses are recovered through timeouts. However, the drawback of Fair Queueing mechanisms is complexity and a larger memory footprint compared to AQM. We also point out the fact that Byte FIFO is a simple scheme and results in very good performance. SPA, our proposal, strikes a compromise between simplicity and the memory footprint of a regular AQM policy. It achieves the low delays of CoDel and performance similar to Fair Queueing schemes. It matches high throughput and low delays required for interactive applications that cannot afford to wait for a 3 s timeout caused by a SYN loss.

Acknowledgments

This work has been partially supported by the French Ministry of Research project PERSYVAL-Lab under contract ANR-11-LABX-0025-01.

References

[1] P. Anelli, F. Harivelo, and R. Lorion, "TCP SYN Protection: An Evaluation," in *Proc. Eleventh International Conference on Networks. ICN'12*, Feb. 2012.

[2] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, pp. 20–34, May 2012.

[3] T. Hiland-Jrgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-CoDel." IETF Internet draft, March 2014. draft-hoeiland-joergensen-aqm-fq-codel-00.

[4] R. Pan, P. Natarajan, F. Baker, and G. White, "PIE: a Lightweight Control Scheme to Address the Bufferbloat Problem," Internet-Draft draft-ietf-aqm-pie-00, October 2014.

[5] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A First Look at Traffic on Smartphone," in *Internet Measurement Conference*, (New York, NY, USA), ACM, 2010.

[6] N. Kuhn, E. Lochin, and O. Mehani, "Revisiting Old Friends: is CoDel Really Achieving What RED Cannot?," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing*, ACM, Aug. 2014.

[7] M. Mellia and H. Zhang, "TCP Model for Short Lived Flows," *IEEE Communications Letters*, vol. 6, pp. 85–87, Feb. 2002.

[8] D. Ciullo, M. Mellia, and M. Meo, "Two Schemes to Reduce Latency in Short Lived TCP Flows," *IEEE Communications Letters*, vol. 13, no. 10, pp. 806–808, 2009.

[9] D. Damjanovic, P. Gschwandtner, and M. Welzl, "Why Is This Web Page Coming Up so Slow? Investigating the Loss of SYN Packets," in *Networking 2009* (D. Damjanovic, P. Gschwandtner, and M. Welzl, eds.), (Berlin, Heidelberg), pp. 895–906, 2009.

[10] A. Vulimiri, O. Michel, P. B. Godfrey, and S. Shenker, "More Is Less: Reducing Latency via Redundancy ," in *Proc. 11th ACM Workshop on Hot Topics in Networks*, (New York, New York, USA), pp. 13–18, ACM Press, 2012.

[11] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing Web Latency: the Virtue of Gentle Aggression," in *Proceedings of the ACM SIGCOMM 2013 Conference*, ACM Press, Aug. 2013.

[12] "The Wonder Shaper." http://lartc.org/wondershaper/.

[13] "DD-WRT." https://www.dd-wrt.com/site/.

[14] B. Turner, "Has AT&T Wireless Data Congestion Been Self-Inflicted?." http://blogs.broughturner.com/2009/10/is-att-wireless-data-congestion-selfinflicted.html.

[15] J. Gettys, "Bufferbloat: Dark Buffers in the Internet," *IEEE Internet Computing*, vol. 15, pp. 96–96, May 2011.

[16] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling Bufferbloat in 3G/4G Networks," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, (New York, NY, USA), pp. 329–342, ACM, 2012.

[17] C. Staff, "Bufferbloat: What's Wrong with the Internet?," *Communications of the ACM*, vol. 55, no. 2, pp. 40–47, 2012. A discussion with Vint Cerf, Van Jacobson, Nick Weaver, and Jim Gettys.

[18] M. Allman, "Comments on Bufferbloat," *ACM SIGCOMM Computer Communication Review*, January 2013.

[19] M. Heusse, S. A. Merritt, T. X. Brown, and A. Duda, "Two-way TCP Connections: Old Problem, New Insight," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 5–15, 2011.

[20] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, no. 5, pp. 20–34, 2012.

[21] R. Pan *et al.*, "PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem," in *IEEE HPSR 2013, Taipei, Taiwan, July 8-11, 2013*, pp. 148–155, 2013.

[22] P. McKenney, "Stochastic Fairness Queueing," *Proceedings of IEEE INFOCOM*, 1990.

[23] J. Schwardmann, D. Wagner, and M. Khlewind, "Evaluation of ARED, CoDel, and PIE," in *Advances in Communication Networking*, vol. 8846, pp. 185–191, Springer, 2014.

[24] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management," tech. rep., Aug. 2001.

[25] N. Khademi, D. Ros, and M. Welzl, "The New AQM Kids on the Block: An Experimental Evaluation of CoDel and PIE," in *2014 Proceedings IEEE INFOCOM Workshop, Toronto, Canada*, pp. 85–90, 2014.

[26] "FreeBSD SYN Cache Source." sys/netinet/tcp_syncache.c. Accessed on: FreeBSD-9.3.

[27] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP Latency," in *Proc. INFOCOM 2003*, pp. 1742–1751 vol.3, IEEE, 2000.

[28] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, pp. 67–82, July 1997.

[29] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," *SIGCOMM Comput. Commun. Rev.*, vol. 28, pp. 303–314, Oct. 1998.

[30] "Linux Traffic Control." http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html. Accessed: 2015-03-18.

[31] "IPMT Test Suite." http://ipmt.forge.imag.fr.

[32] "OECD Broadband Portal." https://www.oecd.org/internet/broadband/oecdbroadbandportal.htm.

[33] V. Bajpai, S. J. Eravuchira, and J. Schönwälder, "Dissecting last-mile latency characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 47, pp. 25–34, Oct. 2017.

[34] T. Braud, M. Heusse, and A. Duda, "TCP over Large Buffers: When Adding Traffic Improves Latency," in *Proc. 26th International Teletraffic Congress (ITC)*, pp. 1–8, Sept 2014.