Policy-oriented AQM Steering

Roland Bless, Mario Hock, Martina Zitterbart
Karlsruhe Institute of Technology
Karlsruhe, Germany
E-Mail: firstname.lastname@kit.edu

***Abstract*—Detecting and handling network congestion in the Internet has, again, become a vital area of research. The provisioning of low latency together with high throughput is of particular interest due to the current mix of applications running in the Internet. Active Queue Management (AQM) mechanisms come with the promise of reducing queuing delays. They, however, may adversely affect throughput and network utilization and have proven to be difficult to configure. More recent AQMs, such as CoDel, PIE, and GSP are easier to configure but work with a fixed target delay setpoint. Depending on the traffic the same setpoint value can result either in unnecessary large delays or under-utilization of the link. *Policy-oriented AQM Steering* automatically adapts the target delay setpoint to the current traffic situation, in order to fulfill a given quality-of-service policy. Such a policy consists of a *utilization goal* and an *upper delay bound*. This improves AQM performance with varying traffic situations and makes the impact of deploying an AQM predictable. A prototypical implementation of AQM Steering for GSP showed its performance advantages compared to static AQM variants at speeds of 10 Gbit/s and 1 Gbit/s.**

I. Introduction

In the last years, the reduction of latency in the Internet has become an increasingly important topic. Applications that especially benefit from low latencies are world-wide web applications due to their transactional character as well as interactive real-time applications such as Voice-over-IP or online games. However, in the current Internet such traffic is mixed with longer lasting and large data volume flows like video streams and downloads. This can cause bandwidth bottlenecks – often located at the consumer edge. As a result packets will queue up in buffers at these bottlenecks, causing an increased end-to-end delay. The "bufferbloat" studies [1] revealed that there exist several places where significantly large buffers are present and that they tend to get filled by TCP's loss-based greedy congestion control strategy. The resulting queuing delay contributes significantly to the end-to-end delay. In extreme cases, end-to-end delay can increase up to several seconds, resulting in poor TCP performance and unusable delay-sensitive applications.

An outcome of the fight against bufferbloat was to revive the use of *Active Queue Management (AQM)* in order to reduce queuing delay. Use of AQM provides several benefits, including enabling Explicit Congestion Notification [2]. Earlier efforts to deploy AQM suffered from their difficult configuration (i.e., several parameters needed to be set and their impact on performance was not obvious) and often negative impact on network utilization (different kinds of traffic required a different set of parameter settings to achieve a good performance). Newer AQMs such as CoDel [3], PIE [4], and GSP [5] are simpler to configure and have a target setpoint that corresponds to the permitted delay limit.

However, the problem of a static configuration remains since the resulting performance depends on the respective traffic. Consequently, the chosen setting can lead to sub-optimal performance [6], e.g., either a too low link utilization or a too high queuing delay. For example, CoDel uses a fixed target of 5 ms by default, which may be too low in some situations: if only a few flows traverse the bottleneck, link utilization is also low and could be increased by permitting a higher target delay. In other situations even lower targets are possible. This means that AQMs with fixed target setpoints cannot sufficiently adapt to the current traffic situation.

The goal of *Policy-oriented AQM Steering* is to provide an automatic adaptation, i.e., adjusting the target delay setpoint to the current traffic situation. Therefore, the AQM mechanism is controlled within given bounds that are set by a provider policy: a *lower bound for link utilization* and an *upper bound for a queuing delay target*.

AQM Steering works a on a different time-scale than AQM auto-tuning. Moreover, it is not integrated into an AQM itself. Instead, it operates as an additional control loop outside of the AQM. This way, it can be easily applied to different AQMs.

II. Problem Analysis

A. Queuing Delay and Counter-Measures

As mentioned before, queuing delay often contributes substantially to the overall latency. Thus, a reduction of queuing delay (i.e., buffer occupancy) is one approach to lower end-to-end latency. Buffers in routers or switches are necessary in order to absorb short-term bursts and to keep link utilization high. There have been numerous debates about the right size of buffers in the past [7], [8]. In addition to absorbing short-term bursts, buffers have another effect on TCP performance. A large buffer allows the *congestion windows ($CWnd$)* of the TCP flows to inflate way beyond the *bandwidth delay product (bdp)* without causing packet losses. This creates a so-called *standing queue* within the buffer [3]. If such an inflated $CWnd$ is eventually reduced after a loss, it can still be above the bdp, thereby maintaining full link utilization. With the well-known "1-bdp Rule of Thumb" (buffer size = $1 \cdot bdp$) this is fulfilled in almost any circumstances. However, if many flows share a bottleneck *and* synchronized losses can be avoided, a smaller

$CWnd$ inflation and, thus, a smaller standing queue would suffice to keep the link fully utilized [7].

There are two different approaches to reduce the standing queue, while maintaining a high throughput:

- Use of a different congestion control that avoids to create substantial standing queues and use different backoff strategies, e.g., TCP LoLa [9] or BBR [10], which are currently under development.
- Use of *Active Queue Management* mechanisms. They try to reduce the standing queue by applying a control loop that early discards packets while retaining the buffer's capability to absorb short-term bursts. Furthermore, AQM mechanisms can support *desynchronization* of packet losses among concurrent TCP flows.

Note that using smaller tail-drop buffers is not a viable option. They lead to lower delays, but also to lower utilization: A small tail-drop buffer cannot compensate for short-term bursts and leads to synchronized packet losses. Both lead to strong backoff reactions of the TCP flows, which reduce their congestion windows way below the necessary size of $1 \cdot bdp$.

This paper focuses on Active Queue Management and assumes that currently used congestion controls, such as TCP Reno, CUBIC TCP or Compound TCP are in place.

B. Room for Improvement of Current AQMs

In contrast to earlier AQM approaches [11] newer AQMs such as CoDel, PIE, and GSP explicitly distinguish short-term bursts from standing queues and thus have a built-in burst tolerance in order to avoid unnecessary packet drops that would decrease the throughput. Moreover, earlier AQM approaches possessed several parameters that needed to be configured. The influence of the parameter setting on the achieved network utilization was often not obvious. Moreover, different traffic types required different parameter settings to achieve the best performance, i.e., they were not "self-tuning" in this respect. This turned out to be a major obstacle for their deployment [12].

Thus, newer AQMs had the objective of being usable across a wider range of scenarios without the need to adapt AQM parameters. CoDel even tries to be "parameterless for normal operation, with no knobs for operators, users, or implementers to adjust", by setting the default target value to 5 ms (5% of a 100 ms measurement interval). Nevertheless, these AQMs still have a configurable target setpoint that corresponds to the permitted delay limit. This target setpoint often relates to an internal threshold that triggers packet drops.

Even though newer AQMs are better in adapting to different traffic scenarios, they still possess their configurable but fixed target setpoint. This creates two-sided drawbacks, depending on the traffic situation, which is mainly characterized by the number of *dominant* flows at the bottleneck, i.e., flows that contribute substantially to the overall in-flight data.

- *Unnecessary high delay* – In case the number of dominant flows traversing the bottleneck is large enough, the AQM can enforce its delay limit while full link utilization can be achieved, due to good loss desynchronization. However, the delay target may be excessively high (cf. Fig. 1a). It could be set to a lower value without sacrificing utilization (as in Fig. 1b).
- *Under-utilization* – In case the number of dominant flows traversing the bottleneck is low (e.g., < 10) or they are having a large RTT, the AQM cannot achieve full link utilization (see Fig. 1c). The reason is the multiplicative decrease backoff of the current TCP congestion controls. With only a few dominant flows or high RTT flows at the bottleneck, the amount of inflight data can easily fall below the bdp. In this case, a higher delay target would maintain a good link utilization (see Fig. 1d).

For the transfer of scientific data, for example, it is a typical pattern that a low number of high volume flows can appear (and disappear) as dominant flows at a bottleneck, at any time. Usually they last for a long time, e.g., hours. Thus, the traffic situation at the bottleneck is significantly changed and a bottleneck (with a fixed setpoint AQM) could fall from an unnecessary high delay to under-utilization.

The goal of Policy-oriented AQM Steering is to find the best trade-off by automatically adjusting the target setpoint within given performance bounds that are specified by a provider *policy*: a *lower bound for link utilization (u_{low})* and an *upper bound for a queuing delay target ($target_{max}$)*.

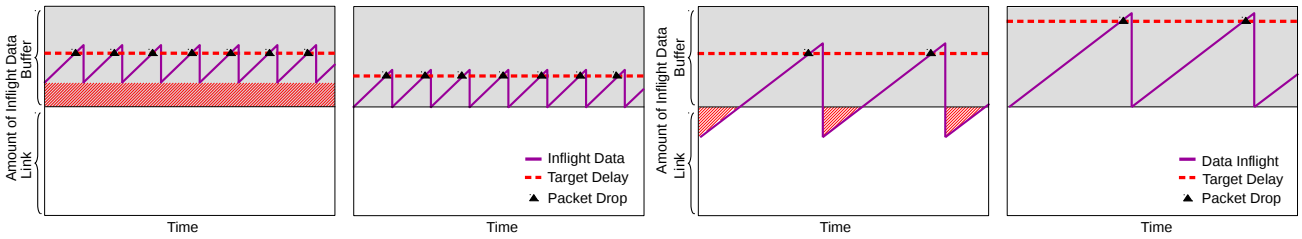
III. Design of AQM Steering

The basic principle of Policy-oriented AQM Steering is to observe how well the momentarily applied target setpoint works with the current traffic situation. If the setpoint is larger than necessary, it can be decreased without violating the lower bound for link utilization (u_{low}). If the setpoint is too small to fulfill this bound, the setpoint has to be raised, as long as the upper bound ($target_{max}$) is not reached. In order to assess the impact of the current target setpoint, the control loop of AQM Steering works outside of the AQM control loop and on a different timescale.

Fig. 2 shows the interplay of the different control loops that interact with each other. The TCP congestion control actually controls the load on the network by reacting on congestion signals (usually packet loss or ECN markings). The AQM tries to find the right amount of congestion signals to emit, in order to effectively control the queue. For this, the reaction of the flows on the congestion signals has to be constantly monitored by the AQM. An important property of this interplay is that it takes at least one RTT for the congestion control to react. As soon as the AQM effectively controls the queue, AQM Steering can determine how well the target setpoint works for the current traffic situation by getting actual values for queuing delay and throughput. It also gets notified of certain events such as packet drops and then determines whether an adjustment is necessary to fulfill the given policy.

A. How can AQM Steering detect when to react?

Three different states have to be distinguished: 1) Link is no bottleneck 2) The AQM is still adjusting to the current load



(a) Target too high – unnecessary high delay (b) Adapted target – Lower delay (c) Target too low – Under-utilization (d) Adapted target – Full utilization

Fig. 1. Sketches illustrating two-sided drawbacks of AQMs with fixed target delay values

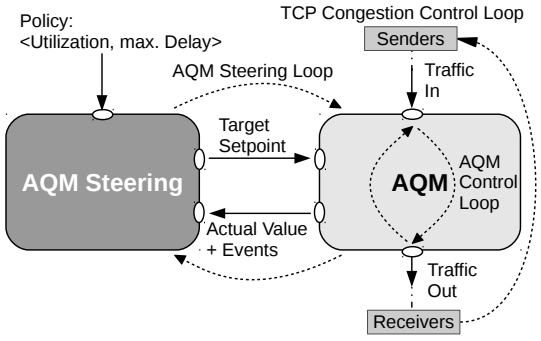


Fig. 2. Interaction of AQM Steering, AQM, and TCP Congestion Control

3) The AQM effectively controls the queue.

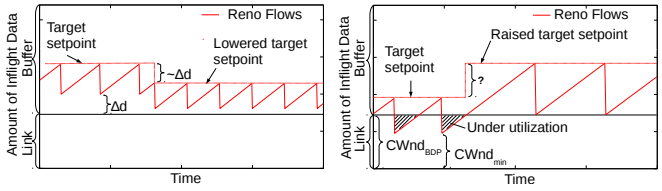
If the queue is persistently above the target, the AQM has not yet adapted to the traffic. In this case it is not expedient to change the target. After the AQM control loop has found a dropping rate that is suitable to effectively control the traffic, the queue will be fluctuating around the target. Now, AQM Steering can assess the impact of the current target setpoint. If the AQM, in contrast, does not drop any packets, the link is no bottleneck, i.e., the queue length is persistently below the target (except for bursts that are ignored by the burst protection of the AQM). Table I summarizes states, causes, and needs for action of AQM Steering.

TABLE I
AQM STEERING – STATES AND NEEDS FOR ACTION

State of Queue	Cause	Adaptation of Target
(1) persistently below target	no bottleneck, no AQM action	not necessary
(2) persistently above target	bottleneck, traffic source(s) did not respond (yet)	not useful
(3) fluctuating around target	bottleneck, AQM active	possible

B. How does AQM Steering determine a new target setpoint?

Actually, two different strategies are required: one for lowering and one for raising the target setpoint. The reason is that one can use measured queue parameters to determine how much reduction of the target is required whereas it is impossible to calculate a required increment in case of under-utilization. Both cases are sketched in Fig. 3 and will be explained in the following (the congestion window increment is in reality sub-linear due to the increasing queuing delay).



(a) Lowering the target setpoint (b) Raising the target setpoint

Fig. 3. Target adaptation problems

1) Lowering the Target Setpoint: If the target setpoint is too high, as shown in Fig. 3a, a persistent minimum standing queue can be measured that does not dissipate even after packet drops. Thus, AQM Steering can determine the minimal queuing delay Δd and reduce the target setpoint by approximately this amount. The key point behind this strategy is that the reaction of TCP flows on a congestion signal depends only marginally on the actual value of the target setpoint. This means that even with the lower setpoint, the queue will not underflow after a congestion signal. Thus, the queuing delay is reduced without harming throughput.

However, if multiple flows are present at a bottleneck, their $CWnd_i$ differ in size and the effect of the individual backoff of each flow might be different. A single Δd as sketched in the simplified exemplary situation shown in Fig. 3a is therefore not sufficient: Δd will actually vary. Thus, an average $\overline{\Delta d}$ of measured minima Δd_i and their variance (σ_n) are calculated. The new target setpoint is calculated as follows:

$$target_{new} = \min\left(target_{old} - \overline{\Delta d} + \gamma\sigma_n, target_{old}\right),$$

with $\sigma_n = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\Delta d_i - \overline{\Delta d})^2}$. $\gamma > 1$ defines a protective margin so that an adaptation does not occur too often if the fluctuation is high. A smaller γ leads to a smaller target value and thus more likely to under-utilization.

2) Raising the Target Setpoint: If the target setpoint is too low to achieve full utilization, it is impossible to calculate the required increment for the target setpoint, because there is no directly usable correlation without knowledge of the flow's bdp, that depends on its RTT. Further dependencies are the backoff factor β (which is different for CUBIC TCP and TCP Reno) and the number of flows. This knowledge is

not typically available at the intermediate system that operates the AQM. Therefore, an approach for a stepwise increment is used.

In order to quickly restore a reasonable throughput when under-utilization has been detected, the target setpoint is restored to a reasonable default value ($thresh_{min,up}$). Since one goal of the AQMs that use a fixed setpoint is to provide a reasonable default value, we recommend to use their default value as $thresh_{min,up}$. If the under-utilization persists, the setpoint is multiplicatively increased (e.g., by a factor 2) until the lower utilization bound u_{low} is fulfilled or the upper limit for the setpoint $target_{max}$ is reached:

$$target_{new} = \min\Big(\max\left(target_{old} \cdot 2, thresh_{min,up}\right), target_{max}\Big)$$

The adaptation happens only if a *short-term smoothed value* as well as a *longer-term smoothed value* of the measured data rate is below the lower utilization bound u_{low}. The short-term smoothed value is simply the mean data rate measured since the last packet drop. This value then contributes to a time-dependent longer-term smoothed value. Further details are given in Sec. IV.

C. Policy Option: Under-utilization

Policy-oriented AQM Steering also allows to define an upper bound u_{target} on the utilization with $u_{low} \leq u_{target} \leq 100\%$. The advantage of using u_{target} is that queuing delay is avoided while there is still a lot of room for short-lived flows as well as other more bursty short-lived traffic. Some providers, for example, avoid utilizations above 50% in order to provide enough protection capacity.

To achieve this goal it may be necessary to discard packets earlier, even before any packets queue up. To achieve this, AQM Steering can optionally switch the AQM from the physical queue to a virtual queue. For this a virtual egress rate $rate_{virtual}$ (with $rate_{virtual} < rate_{physical}$) is defined. The virtual queue tracks the notional buffer utilization that would have built up if the egress link had a data rate of $rate_{virtual}$. The switch to the virtual queue is performed as soon as the target setpoint has reached the minimum value and the current utilization u is still larger than u_{target}. This way, the burst tolerance and loss desynchronization provided by an AQM can still be used, even without any physical queue.

It has to be noted that the virtual queue is just a passively computed number, i.e., the traffic is *not* shaped to the virtual egress rate, since this would induce real queuing delay. Therefore, the link utilization can be above $rate_{virtual}$ for short periods of time. Still, the average link utilization will be below or equal to $rate_{virtual}$. Otherwise the virtual queue would increase indefinitely, which is prevented by the AQM. Whether the average link utilization is actually equal to $rate_{virtual}$ depends on the traffic and the level of loss synchronization. This means that the AQM Steering control loop is still necessary to keep $u_{low} \leq u \leq u_{target}$. However, on the virtual queue AQM Steering does not control the target setpoint. Instead, it adjusts $rate_{virtual}$ in the range of $u_{target} \cdot rate_{physical} \leq rate_{virtual} < rate_{physical}$. If $rate_{virtual} = rate_{physical}$ is necessary to avoid $u < u_{low}$, the AQM is seamlessly switched back to the physical queue. More details are provided in Sec. IV.

D. Discussion

AQM Steering cannot always achieve u_{low}. If the link is no bottleneck, it will inevitably be underutilized. Also, a larger standing queue than allowed by $target_{max}$ might be necessary to attain u_{low}. Hence, $target_{max}$ can be understood as maximal delay one is willing to trade for higher throughput. Often large tail-drop buffers are deployed to get a high throughput at the cost of delay. When converged, the link utilization of AQM Steering will be at least $\min\Big(u_{low}, thr_{td}(target_{max})\Big)$, with $thr_{td}(\ldots)$ being the throughput of a tail-drop buffer of the given size would achieve with the same traffic. Due to the burst protection and loss desynchronization provided by an AQM, the throughput of AQM Steering can actually be larger than $thr_{td}(target_{max})$.

During the convergence of the AQM Steering control loop, throughput can be below $thr_{td}(target_{max})$. Therefore, AQM Steering is tuned to quickly converge towards larger target setpoints (i.e., improving link utilization). Reducing the setpoint (i.e., lowering delay) is performed more conservatively.

IV. Implementation

We chose to use the GSP AQM [13], [5] as basis for our prototype implementation as it is simple to implement and achieves comparable results to CoDel and PIE. GSP's target setpoint is the packet queuing delay in form of a *threshold*. If the *threshold* is exceeded (the sojourn time of the last dequeued packet was larger than threshold) on packet arrival, it immediately discards the arriving packet. GSP then pauses discarding for a time span (called *interval*) allow the congestion control to react on the drop. The *interval* is adapted according to the situation, i.e., if the congestion control reaction was not effective enough to let the queuing delay fall below the threshold, the interval becomes shorter, so that GSP drops more aggressively.

Due to performance and simplicity reasons, AQM Steering was integrated into the GSP implementation, although the general concept allows for a more modular and separated realization. The threshold adaptation of *GSP with AQM Steering (GSP-AS)* is hooked-in into the threshold exceeded event, i.e., $t_{sojourn} > thresh_{curr}$ but is carried out before a packet drop is performed. This way, an increase of the threshold will defer a packet drop until the increased threshold is exceeded. If the threshold is kept unchanged or lowered, a packet is dropped as usual.

Table II shows different variables that are also used in the following description. Policy-oriented AQM Steering allows to set $thresh_{max}$ as well as $rate_{target,min}$ as parameters. Optionally, a third parameter $rate_{target}$ can be set if the desired operational mode is under-utilization. The short-term link utilization is calculated by $rate_{bd} = \sum_{i=0}^{n} packet_{t_i}.size/(t - t_0)$,

TABLE II
SELECTED VARIABLES

Name	Default	Explanation
$thresh_{max}$ *)	configurable	upper bound for the target setpoint
$thresh_{min}$	0,2 ms	lower bound for the target setpoint
$rate_{bd}$	–	data rate since last packet discard
$rate_{bd,avg}$	–	longer-term smoothed data rate
$rate_{max}$	–	maximum link speed, corresponds to $u = 100\%$
$rate_{target,min}$ *)	configurable	lower bound on rate, corresponds to u_{low}
$rate_{target}$ *)	optionally configurable	rate that corresponds to target utilization u_{target}

*) policy is expressed by these parameters

where t_0 is the time of last packet discard, $packet_{t_i}$ denotes a packet that arrived at time $t_i \in [t_0, t]$. Based on these values the longer-term $rate_{bd,avg}$ is calculated with the TDRM-UTEMA-CPA smoothing function [14].

Algorithm 1 Lowering the Target Setpoint

```
1: procedure ATGSPINTERVALADAPTATION
2: ...
3: t_rq_min ← min(t_current_sojourn, t_rq_min)
4: ...
5: V, V_avg ← 0                                   ▷ Initialize at start
6: thresh_curr, t_rq_min ← [thresh_min, thresh_max]
7: procedure ATGSPPACKETDISCARD
8:  if t_rq_min < thresh_curr then
9:   Δt_rq_min ← thresh_curr − t_rq_min
10:  Δt_rq_min_avg ←SMOOTHUTEMA(Δt_rq_min, now)
11:  ESTIMATEVARIANCE(Δt_rq_min)
12:  thresh_down ← thresh_curr − (Δt_rq_min_avg + γ√V_avg)
13:  thresh_down ← ⌊thresh_down/thresh_down_min⌋ · thresh_down_min
14:  if thresh_down > 0 and rate_bd,avg > rate_target,min then
15:   thresh_curr ← max(thresh_curr − thresh_down, thresh_min)
16:  t_rq_min ← thresh_curr
17: ...
```

Algorithm 1 shows the pseudocode for lowering the target setpoint. As discussed above, the target setpoint should not be adjusted if the queue is persistently above the target (see Table I). Therefore, the minimum packet sojourn time t_{rq_min} (rq indicates the real queue, vq the virtual queue) that occurs between two packet discards is tracked and the adaptation is only performed if $t_{rq\_min} < thresh_{curr}$ (line 8). This can be done when GSP interval is adapted anyway (see line 3).

As visualized in Fig. 3, the adaptation amount is calculated based on $\Delta t_{rq\_min} = thresh_{curr} - t_{rq\_min}$. This value is smoothed with the UTEMA function [14] and a variance is calculated. The threshold decrement gets rounded to an integral multiple of $thresh_{down\_min}$ (the minimal allowed threshold value, e.g., 0.2 ms) to avoid too small adjustments (see line 13). This also increases the stability for very low thresholds in combination with smoothing of the measured values. To improve the stability of the mechanism, the new threshold is only applied if the longer-term measurement for the data rate $rate_{bd,avg}$ is above the configured lower bound rate $rate_{target,min}$ (line 15).

If a target utilization goal u_{target} is specified, the decrement by $thresh_{down}$ may not be sufficient to get there. Therefore, a step-wise multiplicative reduction is applied until the minimum threshold $thresh_{min}$ is reached (shown in algorithm 3). If a further reduction is required, the AQM Steering switches to operate on the virtual queue.

Algorithm 2 shows the pseudocode for raising the target. A precondition is that the queue has been drained empty after a packet drop (line 3). If the link is no bottleneck, raising the threshold will not increase the utilization. If a drop will not cause the buffer to empty, link utilization already is at 100 %. Thus, increasing the threshold is not necessary. If line 3 is true, the short-term $rate_{bd}$ *and* the longer-term $rate_{bd,avg}$ are checked against the configured minimum $rate_{target,min}$ (u_{low}). If both are below this target, the threshold is raised (line 10). Otherwise, $rate_{bd} > rate_{target,min}$ alone would often trigger too early, whereas $rate_{bd,avg} > rate_{target,min}$ alone could lead to an unnecessary raise, because of the inertia of $rate_{bd,avg}$, i.e., the necessary threshold could have been reached already in the meantime. If the increase conditions are met, the threshold is multiplicatively increased by a factor $\alpha = 2$. However, if the threshold is very low, the reaction on a sudden load change could be too slow. Therefore, the threshold is at least set to $thresh_{min,up} := 0.025 interval_{init}$. We decided to set this value similar to the default GSP parametrization. This way, the throughput with GSP-AS will be at least as high as with regular GSP after an under-utilization has been detected.

Algorithm 2 Raising the Setpoint Target

```
1: procedure ATGSPPACKETARRIVAL
2: ...
3: if (queue empty) and (packet was discarded) then
4:  thresh_up ← true
5: if t_sojourn > thresh_curr and now > timeout_expiry then
6:  rate_bd ← Data rate since last packet discard
7:  if thresh_up = true then
8:   if rate_bd < rate_target,min and rate_bd,avg < rate_target,min then
9:    thresh_curr ← min(max(thresh_curr · α,
10:                          thresh_min,up), thresh_max)
11:  thresh_up ← false
12:  else
13:   rate_bd,avg ← SMOOTHTDRM(rate_bd, now)
14:   GSPPACKETDISCARD
15: ...
```

Algorithm 3 shows the switch to the virtual queue in case the configured target $rate_{target}$ has not been reached by lowering the threshold as shown in algorithm 1. However, the threshold adaptation cannot be applied in the same way as for the real queue, because the virtual queue does not effectively delay packets. In order to maintain the AQM's property of achieving desynchronization, the virtual departure rate $rate_{vq}$ is adapted instead (line 12). Since the flows congestion windows will fluctuate, $rate_{vq}$ will be often higher than $rate_{target}$, but leading to an effective measured $rate_{target}$ due to the oscillations. Therefore, $rate_{vq} \in [rate_{target}, rate_{max}]$. If the average rate $rate_{bd,avg}$ is lower than the target $rate_{target}$

then $rate_{vq}$ will be raised. If $rate_{vq}$ reaches $rate_{max}$ then the operation uses the real queue again. This is shown in algorithm 4.

Algorithm 3 Rate adaptation and switch to virtual queue

```
1: boolean vq_active                                   ▷ Packet discard according to virtual queue
2: procedure ATGSPPACKETDISCARD
3: ...
4:  rate_bd ← Data rate since last packet discard
5:  rate_bd,avg ← SMOOTHTDRM(rate_bd, now)
6:  if rate_bd,avg > rate_target and rate_target < rate_max then
7:   if vq_active = false and rate_bd > rate_target then
8:    thresh_curr ← max(thresh_min, thresh_vq, thresh_curr · 0.75)
9:    if thresh_curr = thresh_vq then
10:     vq_active ← true
11:  if vq_active = true then
12:   rate_vq ← max(rate_target, rate_vq + α(rate_target − rate_bd,avg))
13: ...
```

Algorithm 4 Rate adaptation and change to real queue

```
1: boolean vq_active                                   ▷ Packet discard according to virtual queue
2: procedure ATGSPPACKETARRIVAL
3: ...
4:  if vq_size > thresh_vq and now > timeout_expiry  then
5:   rate_bd ← Data rate since last packet discard
6:   if thresh_up = true then
7:    if rate_bd < rate_target,min and rate_bd,avg < rate_target,min then
8:     rate_vq ← min(rate_max, rate_vq + α(rate_target − rate_bd,avg))
9:     if rate_vq = rate_max then
10:     vq_active ← false
11:   thresh_up ← false
12:  else
13:   rate_bd,avg ← SMOOTHTDRM(rate_bd, now)
14:   GSPPACKETDISCARD
15: ...
```

V. Evaluation

We evaluated Policy-oriented AQM Steering at speeds of 1 Gbit/s and 10 Gbit/s. One objective was to compare the performance of AQM Steering with tail-drop buffers (small and large) and AQM approaches with fixed targets (namely CoDel and GSP). Another objective was to evaluate the adaptivity of AQM Steering with respect to changing traffic situations. The following experiments were performed: Steady state with long-lived flows, steady state with long-lived and short-lived flows, and transition behavior for changing traffic situations. An additional experiment (with $u_{target} = 95\%$) was performed to evaluate the virtual queue feature. Every experiment was repeated in 10 different runs. Due to space restrictions we present only the results of the 10 Gbit/s experiments, the results for the 1 Gbit/s experiments are very similar.

A. Testbed

Fig. 4 shows the configuration of the testbed where the experiments were conducted. Sender and DPDK-based switch were dual processor servers equipped with Intel Xeon E5-2630, the receiver was a dual processor Intel Xeon E5-2640. All servers were equipped with 128 Gbyte RAM and Intel X710 4-port 10 Gbit/s network cards. The Ubuntu 16.04 Linux distribution was used as operating system, Kernel versions 4.9.0 and 4.4.0 were used. Generic Receive Offload and Generic Segmentation Offload were active. The DPDK switch used bursts of 32 packets and had a configured limit for the send and receive ring buffers of 256 packets.

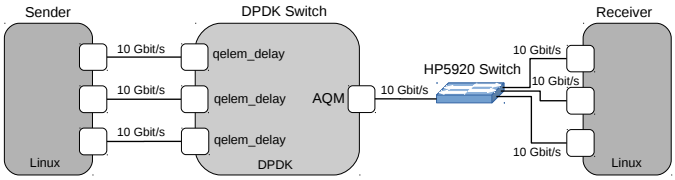


Fig. 4. The Testbed in its 10 Gbit/s Configuration

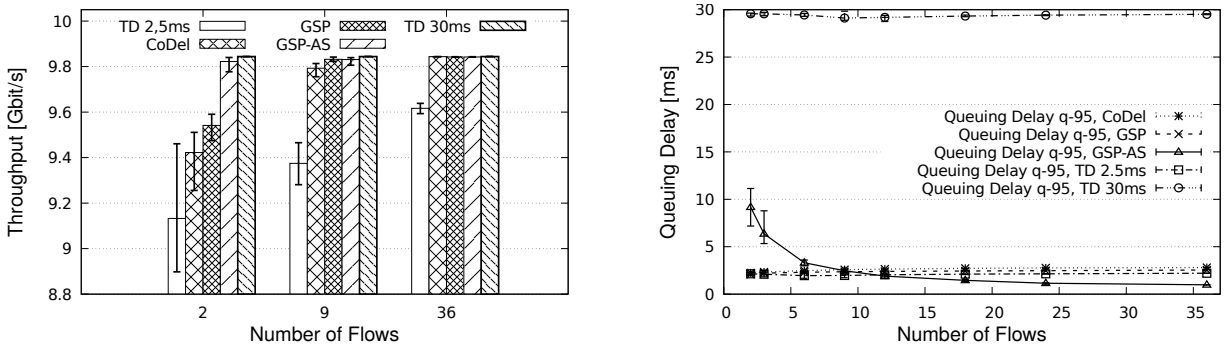
The DPDK-based switch[1] implemented the AQMs CoDel, GSP as well as GSP-AS. It also allows for monitoring internal AQM state such as packet drops and queue length. Moreover, the module `qelem_delay` generates artificial delay, since `netem` is not able to generate reliable behavior at such high speeds. The RTT for all experiments was set to 50 ms. As tools `TCPlog`[2], `CPUnetLOG`[3], and `iperf3` were used.

B. Steady State – Long-lived Flows

This experiment shows that AQM Steering adapts to the traffic situation and achieves higher throughput at the lowest possible queuing delay in comparison to static AQMs or simple tail-drop buffers. Fig. 5 shows GSP-AS in comparison with statically configured GSP and CoDel as well as small (2.5 ms) and large (30 ms) tail-drop (TD) buffers. Static GSP and CoDel were used with target setpoints of 2.5 ms. The upper delay limit $thresh_{max}$ was set to 30 ms, u_{target} was set to 100%, and u_{low} to 99%. CUBIC TCP flows were used (with their standard $\beta_{Cubic} = 0.7$) as traffic load and the number of flows was varied as follows: 2, 3, 6, 9, 12, 18, 24, 36. For clarity Fig. 5a shows only results for 2, 9, and 36 flows.

As expected, the throughput is lower for static AQMs and the small tail-drop buffer if the number of flows is low (in this setting, < 9). This shows that GSP-AS increases the delay in order to achieve higher throughput as desired by the policy. In contrast to the tail-drop buffer, it only increases the delay as necessary, i.e., GSP-AS stays clearly below the allowed 30 ms, as shown in Fig. 5b. This as well as other plots show the average of the 95%-quantile across the 10 runs, the error bars their respective min/max values. The curve for GSP-AS show that if the number of flows becomes higher, GSP-AS can reduce queuing delay even further by lowering the target setpoint. While the static variants of GSP and CoDel also accomplish good throughput values, because of the achieved desynchronization, they stay at their fixed targets of 2.5 ms, whereas GSP-AS can go as low as 0.825 ms.

[1] https://git.scc.kit.edu/TM/DPDK_AQM_Switch (branch: AQM_Steering)
[2] https://git.scc.kit.edu/CPUnetLOG/TCPlog
[3] https://git.scc.kit.edu/CPUnetLOG/CPUnetLOG



(a) Throughput

(b) Queuing Delay

Fig. 5. Throughput and Queuing Delay for Different Numbers of Long-Lived Flows

C. Steady State – Long-lived and Short-lived Flows

This experiment shows that AQM Steering still works if short-lived flows disturb the AQM control: static AQMs loose throughput whereas AQM Steering achieves high throughput at lowest possible queuing delays. Several experiments were made where long-lived TCP flows are disturbed by short-lived TCP transfers of 64 MByte every two seconds. The short-lived flows are usually finished before their slow start phase ends. Although AQMs like CoDel, PIE, and GSP have a kind of burst protection already, short-lived transfers may still decrease the throughput significantly. Fig. 6a shows that static AQM variants perform much worse with a lower number of flows (cf. Fig. 5a). Bursty traffic causes packet drops and it is likely that long-lived flows are affected. The queue within the small tail-drop buffer drains completely between drops as can be seen by the very low throughput values and the practically non-existing delay (Fig. 6b). GSP-AS is able to keep the throughput high (even higher than TD 30 ms), at the cost of increasing the queuing delay. With 36 flows GSP-AS is able to reduce the delay without hurting the throughput.

D. Transition Behavior

This experiment shows how AQM Steering adapts when a sudden change of the traffic situations happens. In contrast to the previous experiments, the traffic situation (here the number of flows) changes during the experiment. At first, only two data flows are started and get to steady state when at $t = 180\,s$ 34 additional flows are started that last for 80 s. Fig. 7a shows how AQM Steering smoothly lowers the target setpoint as the number of flows is higher and raises the target setpoint relatively quickly after the remaining two flows have raised their $CWnd$ up beyond the bdp (that takes 18 s) after the 34 flows have ended at second 260. Conceptually, AQM Steering cannot compensate the time TCP requires to claim free bandwidth after other flows have finished, therefore, the target setpoint is not adapted in absence of any packet drops.

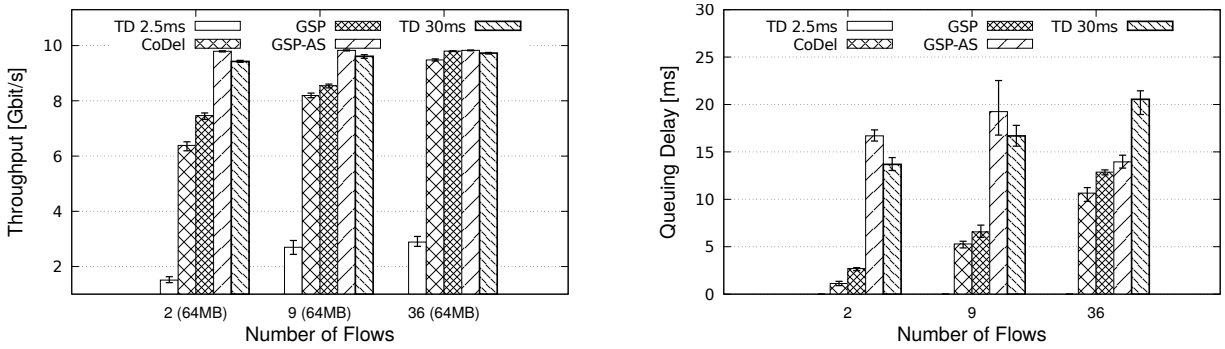
Fig. 7b shows results of an experiment where the situation is repetitively changed before the adaptation of AQM Steering has converged. Every 40 s, 34 additional flows arrive that disappear again after 20 s. During convergence AQM Steering favors high link utilization over a quick reduction of the queuing delay. Indeed, the threshold almost reaches the same upper values as it settled on in the previous experiment. Reduction of the target setpoint, in contrast, is more conservative.

E. Policy Option: Under-utilization

To evaluate the virtual queue feature some experiments were performed with $u_{target} = 95\%$ and $u_{low} = 94\%$. The generated traffic is the same as in Sec. V-B. Fig. 8a shows that the achieved throughput stays within the utilization bounds $[0.94, 0.95]$ (see upper curve and right y-axis). Furthermore, starting at six flows the real queue length is effectively zero (as shown by the queue length curves of the average and the 95% quantile), since the AQM triggers regularly packet discards based on the virtual queue. For a lower number of flows, the virtual queue length fluctuates more (as explained in Sec. II), thus often overshooting into the real queue, as also indicated by the difference of the 95% quantile and the average queue length. Fig. 8b shows the virtual queue length and packet discard actions from a run with 36 concurrent flows. As with the physical queue, the virtual queue length fluctuates around the target. Since the virtual queue is almost never empty, the egress rate is close to $rate_{virtual}$, on average. AQM Steering adjusts $rate_{virtual}$ in order to keep $u_{low} \leq u \leq u_{target}$.

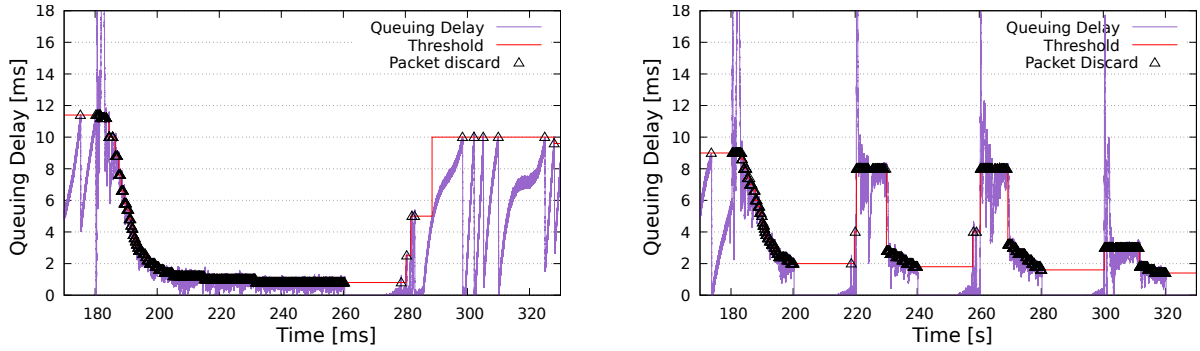
VI. Related Work

The first major AQM mechanism RED faced severe deployment difficulties, due to its sensitivity to parameters, which were hard to tune. Since then auto-tuning has become a standard functionality of subsequent AQMs, like Adaptive RED (ARED) [15], BLUE [16], Optimal Drop-Tail/Optimal BLUE [17]. The latter approach tries to optimally trade-off delay versus throughput based on utility functions. [11] gives an exhaustive survey of existing AQMs. Newer AQMs like CoDel [3], PIE [4], and GSP [5] try to be mostly parameterless, due to inherent auto-tuning and reasonable default values. But due to their fixed delay target setpoint, the performance still depends on the traffic characteristics and is, therefore, hard to predict. Policy-oriented AQM Steering focuses on an automatic adjustment of this parameter, in order give better and more predictable performance.



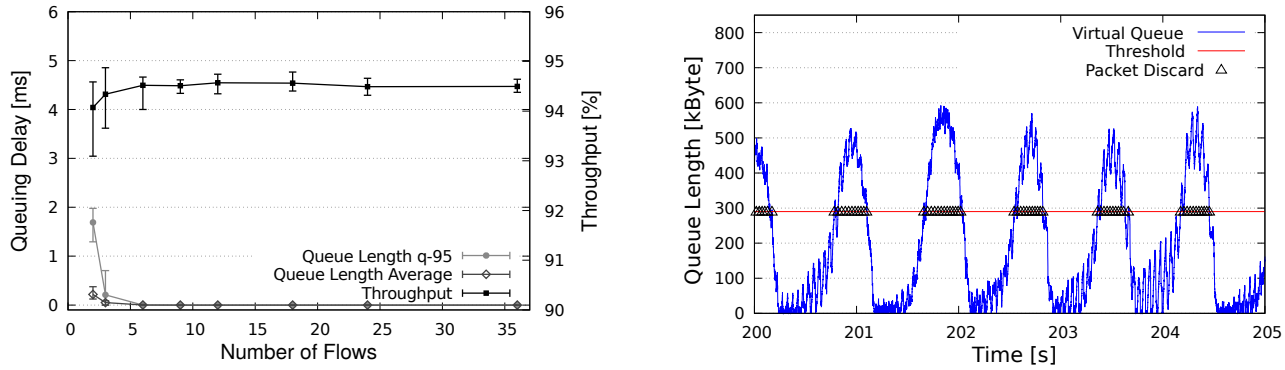
(a) Throughput (b) Queuing Delay

Fig. 6. Throughput and Queuing Delay for Different Numbers of Long-Lived Flows Disturbed by Short Flows (64 MByte)



(a) Slow changes (b) Faster changes

Fig. 7. Transition Behavior



(a) Achieved Throughput and Queuing Delays (b) Length of the Virtual Queue (36 flows)

Fig. 8. Behavior for $u_{target} = 95\%, u_{low} = 94\%$ and upper delay limit 2.5 ms

The work in [6] assesses the operating ranges and the tunability of the AQMs CoDel and PIE. The authors conclude that "manual tuning can hardly be avoided" for some use-cases that lie outside the operating range of the default parameter set. But even for scenarios within the operating ranges, different trade-offs between queuing delay and throughput are possible.

These trade-offs can be tuned by altering the target setpoint. Policy-oriented AQM Steering does exactly this but in an automatic manner. The authors of [6] also found that adapting the update interval λ to the actual RTTs might be useful in some cases. Our mechanism does not tune this parameter, since the RTTs are usually not known by routers/switches. Furthermore, GSP (which our implementation is based upon) does not have such a fixed update interval as CoDel and PIE.

The concept of virtual queues was first introduced as part of [18]. Based on this concept the AQMs AVQ [19] and HULL [20] have been developed. HULL uses so-called phantom queues that simulate queue buildup for a virtual egress link that runs at a fixed fraction of the actual link (e.g., 95 %), with the goal to leave "bandwidth headroom". HULL is designed to be used in conjunction with DCTCP [21]. AVQ simulates a virtual tail-drop queue with a variable virtual

egress rate. The virtual rate is adjusted according to the length of the physical queue, in order to achieve a certain ingress rate ($\leq 100\%$). Our approach also uses a virtual queue, if an upper utilization target is set that cannot be achieved by operating on the physical queue. It further differs from the other approaches by using an AQM in order to achieve a good loss desynchronization in combination with an outer control loop that regulates the virtual egress rate.

VII. Conclusion

Policy-oriented AQM Steering provides an external control loop that dynamically adjusts the target setpoint of newer AQMs. Depending on the traffic this can lead either to lower queuing delays or higher utilization of the bottleneck link. Without AQM Steering, AQMs provide a trade-off between link utilization and delay that is hard to determine, since it changes under different traffic situations. With AQM Steering a simple to grasp policy can be set, consisting of: $\langle u_{low}, target_{max} \rangle$ and optionally u_{target}. This makes the deployment of AQM more predictable and can even improve the performance, e.g., if traffic patterns change over time or are different than expected.

As shown in the evaluation, AQMs can cause a significant drop in link utilization (down to 60 %–80 %) under certain circumstances. Network providers could, therefore, be reluctant to deploy AQMs. In these cases, higher link utilization can be attained at the cost of permitting a larger queuing delay. AQM Steering's policy allows network providers to specify how much queuing delay they are willing to trade for high throughput. But in contrast to large tail-drop buffers (or statically configured AQMs with high target setpoints), AQM Steering only permits these delays when necessary. Otherwise, the delay is reduced to the minimal value that is required to achieve the desired throughput. In addition to that, AQM Steering can optionally switch the AQM to a virtual queue, which allows to specify upper utilization targets. This enables policies that focus on zero queuing delay by enforcing spare capacity. The evaluation has shown that the concept works well for different traffic situations. When traffic patterns change, AQM Steering requires some time to adapt. But due short-term and longer-term smoothing, quickly changing traffic situations do not destabilize the control. Investigation of AQM Steering in more complex scenarios and with different traffic mixes is planned as future work.

Acknowledgment

The authors would like to thank Moritz Kunze for his contributions, thorough and intensive work on the topic, implementation, and evaluation. This work was supported by the bwNET100G+ project, which is funded by the Ministry of Science, Research, and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

References

[1] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Queue*, vol. 9, no. 11, pp. 40:40–40:54, Nov. 2011. [Online]. Available: http://doi.acm.org/10.1145/2063166.2071893

[2] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–63, Sep. 2001.

[3] K. Nichols and V. Jacobson, "Controlling Queue Delay," *Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2208917.2209336

[4] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, July 2013, pp. 148–155.

[5] W. Lautenschlaeger and A. Francini, "Global Synchronization Protection for Bandwidth Sharing TCP Flows in High-Speed Links," in *Proceedings of 16th International Conference on High Performance Switching and Routing (IEEE HPSR 2015)*, Jul. 2015, budapest, Hungary.

[6] N. Kuhn, D. Ros, A. B. Bagayoko, C. Kulatunga, G. Fairhurst, and N. Khademi, "Operating ranges, tunability and performance of CoDel and PIE," *Computer Communications*, vol. 103, no. Supplement C, pp. 74–82, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366416302717

[7] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 281–292, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1030194.1015499

[8] A. Vishwanath, V. Sivaraman, and M. Thottan, "Perspectives on Router Buffer Sizing: Recent Results and Open Problems," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 2, pp. 34–39, Mar. 2009. [Online]. Available: http://doi.acm.org/10.1145/1517480.1517487

[9] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion Control for Low Latencies and High Throughput," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, Oct 2017, pp. 215–.218.

[10] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, no. 5, pp. 50:20–50:53, Oct. 2016.

[11] R. Adams, "Active Queue Management: A Survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1425–1476, Q3 2013.

[12] F. Baker (Ed.) and G. Fairhurst (Ed.), "IETF Recommendations Regarding Active Queue Management," RFC 7567 (Best Current Practice), RFC Editor, Fremont, CA, USA, pp. 1–31, Jul. 2015.

[13] W. Lautenschlaeger, "Global Synchronization Protection for Packet Queues," Internet-Draft draft-lauten-aqm-gsp-03, May 2016, work in progress, https://tools.ietf.org/html/draft-lauten-aqm-gsp-03.

[14] M. Menth and F. Hauser, "On Moving Averages, Histograms, and Time-Dependent Rates for Online Measurement," *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, Apr. 2017, preprint. [Online]. Available: https://atlas.informatik.uni-tuebingen.de/~menth/papers/Menth17c.pdf

[15] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management," AT&T Center for Internet Research at ICSI, Tech. Rep., 2001.

[16] W. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The BLUE Active Queue Management Algorithms," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 513–528, Aug 2002.

[17] R. Stanojević and R. Shorten, "Trading link utilization for queueing delays: An adaptive approach," *Computer Communications*, vol. 33, no. 9, pp. 1108–1121, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366410000897

[18] R. J. Gibbens and F. P. Kelly, "Resource pricing and the evolution of congestion control," *Automatica*, vol. 35, no. 12, pp. 1969–1985, 1999.

[19] S. S. Kunniyur and R. Srikant, "An Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 286–299, Apr. 2004. [Online]. Available: http://dx.doi.org/10.1109/TNET.2004.826291

[20] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228324

[21] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1851275.1851192