Traffic-Driven Implicit Buffer Management– Delay Differentiation Without Traffic Contracts

Martin Karsten
David R. Cheriton School of Computer Science
University of Waterloo
Email: mkarsten@uwaterloo.ca

Daniel S. Berger and Jens Schmitt
Department of Computer Science
University of Kaiserslautern
Email: {berger,jschmitt}@cs.uni-kl.de

***Abstract*—Different network applications have different service preferences regarding packet delay and buffering. Delay management requires scheduling support at routers, which traditionally also requires some form of traffic specification and admission control. In contrast, this paper studies the problem of guaranteeing queueing delay bounds for multiple service classes without traffic contracts and without affecting the throughput rate for each class. A solution to this problem is given by decoupling throughput and delay management via traffic-driven implicit buffer management. Using this concept, the Delay Segment FIFO (DSF) packet scheduler guarantees differentiated delay targets in the presence of unregulated throughput rates. This decoupling represents a modular approach and DSF embodies a small and self-contained feature set. Furthermore, DSF's service model satisfies even a strict interpretation of network neutrality, while effectively guaranteeing delay targets for multiple service classes. DSF's design and service characteristics are analyzed mathematically and validated through simulations.**

I. Introduction

The Internet serves a multitude of diverse applications that have different service requirements. Traditional bulk data transfers seek to optimize their overall throughput. On the other hand, interactive multi-player games or live conferencing systems have stringent end-to-end delay requirements on the order of 50-100ms. A number of applications fall in between, e.g., multimedia streaming requires moderate throughput and has somewhat relaxed end-to-end delay requirements. As an illustrative example, current mobile networks define 13 service classes with end-to-end delay targets of 50ms-300ms [1], i.e., within one order of magnitude. However, the controllable portion of the end-to-end delay is queueing delay. When subtracting speed-of-light propagation latency, the actual queueing delay diversity covers two orders of magnitude.

Traditional approaches to limit and differentiate worst-case queueing delay for different types of applications combine offline or signalled traffic contracts with differentiated scheduling, such that forwarding capacity is allocated to dedicated service classes. As long as traffic arrivals conform to the negotiated traffic specification, the queueing delay is guaranteed by ensuring a particular minimum throughput service. Packet schedulers used in this context include processor sharing and its variants [2], as well as priority scheduling. Thus, delay management is tied to throughput management. However, the exclusive nature of these agreements requires a typically complex policy framework to determine which traffic is eligible for certain service classes. This traffic discrimination might violate a strict interpretation of network neutrality.

An alternative approach is controlling queueing delays without explicit throughput management - by implicitly limiting the effective queue, i.e., buffer, size available for each service class: packets that (will) miss their delay target are discarded. To avoid any throughput bias, service rates must be proportional to arrival rates, while the effective queue size for each class must be automatically adjusted to enforce the respective delay target. This traffic-driven approach provides less direct control, but has a number of practical advantages:

- Without explicit throughput management, sources can pick any service class. This removes the need for a complex policy system determining service class eligibility.
- With unrestricted access to service classes, the network service clearly satisfies the so-called *no paid prioritization* rule, which is one of the three basic rules in the recent FCC Open Internet Order [9].
- Because throughput and delay management are decoupled, the resulting architecture is modular.

The first contribution of this paper is the definition of a service metric *throughput interference index* (TI^2) to quantitatively capture the above objectives for a packet scheduler. The main challenge is devising an efficient packet scheduler that allocates service rates in proportion to arrival rates to achieve a good TI^2 while enforcing per-class queueing delays by dynamically and automatically adjusting the effective queue size for each class. This paper presents the first proper solution to this problem and makes the following contributions:

- The non-technical descriptions of network neutrality are reframed using the more restrictive notion of TI^2.
- Our approach – *delay segmentation* – is shown to achieve a low TI^2, while providing effective delay guarantees.
- A novel multi-class scheduler using delay segmentation is proposed and evaluated. It guarantees delay targets with minimal throughput interference.

The rest of the paper is organized as follows. Section II provides background and the definition of TI^2. Section III discusses related work. Section IV presents the DSF algorithm. The principles behind DSF are analyzed in Section V and it is evaluated using simulations in Section VI. The paper is wrapped up with a brief conclusion in Section VII.

II. Background and Motivation

A. Network Neutrality

The academic discussion of network neutrality spans almost two decades in the legal literature [9], [21], [28], [30]. Yet, from a technical perspective, this discussion has been criticized as being largely oversimplified [7]. In particular, the question which specific packet schedulers and/or service models conform to network neutrality has received little attention in the legal literature. Only FIFO scheduling has been discussed widely and, in fact, many claim that FIFO scheduling is necessary to attain network neutrality [6], [31], [32]. The current network neutrality rules in the USA [9] do not require specific schedulers but instead establish three principles: 1) no blocking, 2) no throttling, 3) no paid prioritization. Further clarification seems necessary [28] to apply these principles. The literature about detecting network neutrality violations typically uses a broader definition. In recent work [26], for example, network neutrality violations are detected by comparing each flow's loss rate to a baseline loss rate, e.g., of some traffic aggregate. The network is neutral, if the difference between each flow and the baseline is sufficiently small. The same principle can be used to derive a metric for quantitatively assessing specific packet schedulers or service models.

B. Throughput Interference Index

We propose a new metric to assess schedulers and service models without the need for a baseline: The throughput interference index TI^2 quantifies the degree by which a scheduler changes the throughput of multiple flows.

Definition 1: The throughput interference index (TI^2) of a rate transformation for n flows, with λ_i^{in} and λ_i^{out} being the input and output rates for flow i, is measured as

$$TI^2 = 1 - \frac{\left(\sum_{i=1}^{n} \frac{\lambda_i^{out}}{\lambda_i^{in}}\right)^2}{n \sum_{i=1}^{n} \left(\frac{\lambda_i^{out}}{\lambda_i^{in}}\right)^2}.$$

Essentially, the TI^2 definition applies Jain's fairness index [15] to relative throughput rates. TI^2 ranges from 0 to $\frac{n-1}{n}$, with values closer to 0 indicating lower scheduler interference with throughput rates. In contrast, higher relative throughput differences result in a larger TI^2. This definition is backwards compatible with previous definitions. Firstly, FIFO indeed has a low TI^2 as shown in the associated technical report [17]. Secondly, this definition incorporates the no blocking and no throttling rules [9] because it ensures that relative service rates correspond to relative arrival rates. Thirdly, this definition incorporates the loss-based metric from recent work [26], because the ratio $\frac{\lambda_i^{out}}{\lambda_i^{in}}$ corresponds to $1 - loss$ for each flow. The analysis of the DSF scheduler proposed in this paper is based on the TI^2 metric.

C. Incentive Compatibility

If all service classes of a multi-class service model are accessible to all traffic without restriction, this trivially satisfies the no paid prioritization rule [9]. However, if a service class is strictly superior to another one in terms of service quality, the resulting race to the top renders the lower class useless. Thus, no service class should be dominated by another one in terms of service quality. The Incentive-Compatible Differentiated Scheduling (ICDS) proposal [18] for multi-class delay differentiation satisfies this requirement. As evident by its name, ICDS identifies the incentive-compatible nature of such a scheduler and presents a simple proof of this property. The same proof can be applied to some of the systems discussed in the next section, as well as the DSF algorithm presented in Section IV. Thus, DSF is also incentive-compatible. Furthermore, ICDS can be expected to have near-zero TI^2, because it is designed to allocate service rates in proportion to arrival rates. However, no practical algorithm is given. In contrast, this paper introduces a specific algorithm, DSF, that also has a very low TI^2.

III. Related Work

The Alternative Best Effort (ABE) scheduler [14] offers a bounded-delay service class ("Green") and a throughput-oriented service class ("Blue"). The Blue class is guaranteed to achieve the same throughput as in the equivalent FIFO system, while the Green class receives priority service whenever possible without violating the throughput guarantee for Blue. However, this priority-based concept is fundamentally limited to two classes. A potential third class cannot receive priority service (in comparison to Blue) and throughput guarantees (in comparison to Green) at the same time. In contrast, DSF supports a general service model with multiple service classes.

Virtually Isolated FIFO Queueing (VIFQ) [16] is a previous attempt to generalize ABE to multiple classes and to implement ICDS [18]. Unfortunately, VIFQ's delay control approach is inherently flawed (cf. Sections V-A and VI-A).

Various schedulers provide non-dominant service differentiation with trade-offs between throughput and delay. The most recent incarnations of this approach are the RD proposal [25] and QJump [12]. RD provides two service classes with a fixed throughput ratio between individual application flows in each class. Appropriate buffer sizing ensures a maximum queueing delay for the delay-oriented class. The design resembles weighted processor sharing with a specific throughput ratio, which would entrench a particular service policy in routers and does not satisfy the objective of a low TI^2. Similarly, QJump couples priority levels (for delay guarantees) with strict limits on the maximum throughput to create trade-offs. Its delay guarantees depend on network parameters, while priorities and throughput limits must be enforced at end hosts. These are unrealistic assumptions for the general Internet. In contrast, DSF is incentive-compatible and makes no such assumptions.

ABE, RD, and QJump all propose penalties for low-delay traffic. This is often justified by the observation that long-term TCP goodput is inversely proportional to the average round-trip delay, which includes the average queueing delay [23]. For example, ABE and RD contain complex rate control mechanisms to prevent TCP-like traffic from gaining an advantage by using the low-delay class. In ABE, the low-delay service class

is penalized using a probabilistic parameter g, which can be computed precisely, only if the round-trip propagation delay of all competing TCP flows is identical and known. In RD, a fixed ratio $k > 1$ is configured as the ratio of the average per-flow rate of the throughput class in relation to the average per-flow rate of the delay class. RD maintains the relative throughput of both classes according to k and the number of flows in each class, which must be estimated. In contrast, DSF is based on a much simpler scheduling algorithm that does not require parameter estimation.

Mathis [22] has questioned this traditional, narrow focus on TCP-friendly rate control. For example, recent alternative TCP rate control algorithms [8] seek to overcome the dependency of TCP's rate on the round-trip delay. In fact, these rate control algorithms demonstrate how a particular service policy in routers would unnecessarily contribute to the ossification of the Internet architecture. Therefore, TCP's current properties should not curtail the search space for packet schedulers without further investigation. DSF only gives delay guarantees (by offering limited queueing space) without prioritizing service. Without prioritization, there is no need to actively penalize low-delay traffic, which simply seeks less queueing space.

Several differentiated service proposals use differentiated buffer management [5], [13]. This segregation of buffers is different from DSF, which manages a single shared buffer. Unlike differentiated services, DSF is incentive-compatible and does not require traffic contracts.

Another approach has recently (re)gained research interest: controlling queueing delay through active queue management (AQM). A prominent example is the CoDel algorithm [24]. CoDel counteracts prolonged periods of high queueing delay by increasing the packet drop rate until the delay reaches a configured value, but tolerates transient phases of higher delay to absorb short-term traffic bursts. This approach is expected to consolidate low delay and high goodput, and indeed, a recent evaluation shows that CoDel achieves the best goodput among several other AQMs [20]. Nevertheless, CoDel does not solve the fundamental trade-off between delay and goodput. In some scenarios, CoDel can increase the file completion time by 42% [20]. In summary, there is no conclusive evidence that AQM can support all desirable delay targets under all circumstances without negative side effects [17].

IV. Delay Segment FIFO (DSF)

Delay Segment FIFO (DSF) is a multi-class packet scheduler that closely approximates the per-flow throughput of FIFO and thus attains a low TI^2. Each service class has a fixed delay target and packets are assumed to carry a service class identifier in their header, for example, in the DiffServ code point [4]. The DSF algorithm applies two key principles, *Delay Discard* and *Queue Segmentation*, to achieve its objectives.

Delay Discard decouples buffer admission control from per-class delay management. This is facilitated by using two separate queue types – one for *slots* that represent a right to service and another one for packets. Each arriving (and accepted) packet creates a corresponding slot that is appended to the global slot queue, while the packet is stamped with its per-class delay deadline and added to a per-class packet queue. For each slot in the slot queue, the service routine processes the corresponding packet queue and discards packets that have missed their deadline. The next packet from this class that satisfies the delay target, is sent. By enforcing the target delay in this way, Delay Discard implicitly adapts the effective buffer size for each service class to its arrival rate. This design is motivated by viewing the buffer demands of a traffic flow as the dual of its delay objective. A traffic flow with average rate r and delay target d can reasonably expect at most a buffer capacity of d/r. The earlier VIFQ proposal [16] is an attempt at a practical implementation of this conceptual design. However, Section V-A presents a Markov chain analysis that shows a fundamental flaw of relying on Delay Discard only. In particular, when the slot of a low-delay class reaches the front of the slot queue for service, the class might not have a packet available that satisfies the delay target, if its traffic rate is relatively low. This results in a high TI^2 in those cases.

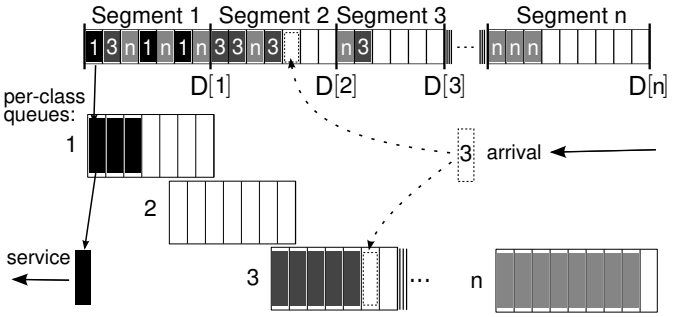


Fig. 1: Delay Segment FIFO

Therefore, DSF introduces *Queue Segmentation* in the buffer organization: each delay target is represented by a corresponding queue segment. The intuition behind this approach is making the system appear as a short virtual FIFO queue for low-delay traffic, while offering a bigger FIFO queue for higher-delay traffic. The key algorithmic change introduced with Queue Segmentation is the last-in-first-out (LIFO) service order between queue segments, which addresses the high throughput interference of Delay Discard. Assume that n delay classes are ordered according to their delay targets. The total buffer space is split into n queue segments. The size of segment $i = 1, \ldots, n$ corresponds to the incremental delay target of class i. In other words, if the link rate is R and the delay targets are $D_1, \ldots, D_n$, then the capacity of each segment i is $S_i = (D_i - D_{i-1}) \cdot R$ (with $D_0 = 0$). Thus, the aggregate capacity of all segments is equivalent to the highest delay target. For each service class a packet queue holds the packets in arrival order, ensuring FIFO service order per class.

A. Algorithm

DSF is illustrated in Figure 1 and specified in Algorithms 1 and 2. Data structures and variables are described in Table I.

Upon arrival, the first non-full segment is determined (Lines 3-5). If successful, a slot is appended to the segment queue

Algorithm 1 DSF Arrival Routine (per packet)

```
1: p ← received packet
2: c ← class(p)
3: i ← 1
4: while i ≤ c and squeue[i].full() do i ← i + 1
5: if i ≤ c then
6:     squeue[i].push(c, size(p))
7:     buffer[c] ← buffer[c] + size(p)
8: while pqueue[c].size() + size(p) > buffer[c] do
9:     pqueue[c].pop()
10: pqueue[c].push(p, now() +D[c])
```

Algorithm 2 DSF Service Routine (loop)

```
1: i ← 1
2: while squeue[i].empty() do i ← i + 1
3: {c, s} ← squeue[i].pop()
4: buffer[c] ← buffer[c] - s
5: service[c] ← service[c] + s
6: while service[c] ≥ 0 and pqueue[c].nonempty() do
7:     {p, d} ← pqueue[c].pop()
8:     if now() < d then
9:         service[c] ← service[c] - size(p)
10:        send(p)
```

TABLE I: DSF Variables and Routines

Name	Index	Explanation
squeue[]	segment	slot queue: {class, size}
pqueue[]	class	packet queue: {packet, deadline}
buffer[]	class	buffer size (sum of waiting slots)
service[]	class	service credit/debit
D[]	class	delay target
now()	N/A	current time

(Line 6) and the per-class buffer size is increased (Line 7). Lines 8 and 9 implement a *Drop Front* mechanism per class, which is discussed in the next paragraph. Last, the arrived packet is appended to the per-class packet queue (Line 10).

At service time, if a non-empty segment is found (Lines 1,2), the first slot is removed (Lines 3,4) and the corresponding service class is given the appropriate service credit (Line 5). The essence of the Delay Discard function is found in Lines 6-10 of the service routine, which trivially guarantees that delay targets are met. As long as there is a positive service credit and available packets, a packet from the per-class packet queue is retrieved (Lines 6,7). If it satisfies the delay target (Line 8), it is transmitted (Line 9,10). Variable packet sizes are handled through the service credit counter. Because each packet is sent in entirety, a class might temporarily receive more service than it is entitled to. However, the service error is limited to one packet per class and corrected over time. Also, the delay test of DSF operates on packet serialization start times to avoid favouring small packets. This introduces a delay error in the amount of at most one maximum packet size per class.

Using only Delay Discard and Queue Segmentation would not limit the total amount of packet buffer needed and would increase the amount of processing in the service routine to discard packets that never had a chance to meet their delay target. Therefore, DSF uses an additional *Drop Front* mechanism in the arrival routine. There is no point in storing more packets in the packet queue than the total amount of sending rights for a traffic class given by the sum of its slot sizes. Lines 8,9 of the arrival routine thus limit the number of packets in the packet queue and also ensure that only the most recently arrived packets are kept, because those packets have the best chance of meeting their delay target.

B. Remarks

The DSF algorithm is *traffic-driven*, because the effective buffer size for each service class is determined by its arrival process in combination with the fixed delay target. Buffer management is *implicit*, because the buffer size is not computed explicitly before packet admission control, but is derived indirectly from admitted slots.

DSF is inherently a single-node technique. Therefore, it can be deployed incrementally, for example at critical bottlenecks only. The flip side is that a flow traversing multiple bottlenecks might be subject to multiples of the delay target. However, the number of bottlenecks is usually small. DSF can support very aggressive delay targets, as shown in Section VI. In a system without Delay Discard, late packets are forwarded and increase downstream load, despite being ultimately useless.

C. Overhead and Complexity

The computational overhead of the DSF algorithm is small and limited. The only data structures that are used for DSF are plain FIFO queues storing either slots or packets. The only numerical operations are addition and subtraction. There is no sorting or other rearrangement of information that would impose computational complexity. The segment search in the arrival routine (Lines 3-5) and the service routine (Lines 1,2) can be implemented using bitstrings, along with the constant-time *find-first-set* operation that is available on modern hardware. The overhead of the arrival routine is proportional to the size of the arriving packet divided by the minimum packet size. Similarly, the total overhead of the service routine is proportional to the amount of sending rights stored in the slot segments divided by the minimum packet size. Therefore, DSF has amortized constant complexity per packet. On the other hand, the worst-case overhead between two consecutive service invocations might be increased by a series of packets that cannot meet their delay target in the loop in Line 6 of the service routine. However, the Drop Front function in the arrival routine limits the size of the overall packet buffer and furthermore, should keep the number of packets that miss their deadline small. Therefore, it is expected that the length of such service drop episodes is limited in practice. For example, over all the experiments reported in Section VI, the average length of service drop episodes is 1.3 packets with a standard

deviation of 1.18. Additional improvements are possible by exploiting parallel hardware, which is abundantly available.

V. Analysis

This section presents analytical models for the two key principles of DSF: Delay Discard and Queue Segmentation. To keep the models tractable, they are based on fixed-size packets and a discrete time model with each slot corresponding to the service time of one packet. Sources are fully described by their arrival rate (i.e, no correlations are assumed); no single source has more than one packet arriving in a single time slot, which essentially means geometrically distributed packet inter-arrival times. Only two delay classes are modelled with Class i having a delay target of N_i time slots. Consequently, the total buffer capacity is $N = N_2$ slots. The arrival rates for Class 1 and 2 are denoted as r_1 and r_2.

A. Delay Discard Only

The goal of this section is to find a closed-form expression for the TI^2 of Delay Discard (DD). Under the above assumptions, a global Markov chain could be created that closely tracks the dynamics of the overall system. The state of the system would be captured by a vector (of length N) whose entries could take three values: (1) *assigned* slot (reserved for an in-time packet of Class 1 or for a Class 2 packet), (2) *uncertain* slot (reserved for Class 1, but packet that created it is late) (3), *empty* slot. The Markov chain could be used to determine the probability of an uncertain slot arriving at the head of the queue without a suitable Class 1 packet available to use it. Such a slot is called *expired*. We have experimented with this model, but deem it intractable, because of its complex structure, as well as a dramatic state space explosion, which also prohibits a numerical solution.

1) Decoupling Approach: Since the global Markov chain is not a feasible choice for the analysis of DD, an approximation is used that assumes a decoupling of the arrival and slot generation processes. The system is modelled as two Markov chains that interact with each other in a simple way. In particular, one Markov chain is used to model the slot generation process by tracking the total queue length to derive the drop rate due to a full slot queue. In addition, it is used to determine the *overload* probability, i.e., the probability that the number of service slots in the slot queue exceeds N_1, the delay target of Class 1. The drop rate computed from this Markov chain can be used to drive another Markov chain, which keeps track of the position of the first uncertain slot in the relevant buffer spaces ranging from $1, \ldots, N_1$. This Markov chain is conditioned on the system being in overload, which provides another link between the two Markov chains.

Based on the two Markov chains, the steady-state probability for an expired slot of Class 1 is obtained as

$$\begin{aligned} p_{E_1} &= \lim_{t\to\infty} \mathbb{P}\left(E_1(t)\right) \\ &= \lim_{t\to\infty} \mathbb{P}\left(E_1(t) \mid O_1(t-N_1)\right) \mathbb{P}\left(O_1(t-N_1)\right) \\ &= p_{E_1|O_1} \cdot p_{O_1}\,, \end{aligned}$$

where $E_1(t)$ denotes the event that a service slot of Class 1 expires at time t, and $O_1(t-N_1)$ denotes the event that the system was in overload at time $t-N_1$. Note that the law of total probability is correctly applied in the second line as $O_1(t-N_1)$ is necessary for $E_1(t)$, i.e., $E_1(t) \subset O_1(t-N_1)$. The steady-state probabilities $p_{E_1|O_1} = lim_{t\to\infty}\mathbb{P}\left(E_1(t) \mid O_1(t-N_1)\right)$ and $p_{O_1} = \lim_{t\to\infty} \mathbb{P}\left(O_1(t-N_1)\right)$ are calculated from the Markov chains for the uncertain slot position and for the total queue length, respectively.

2) Markov Chain for Total Queue Length: This first Markov chain models the total queue length oblivious to traffic classes. Under the given assumptions, this is very similar to a $Geo/D/1/N$ queue [11] with the slight difference that the arrival process is a superposition of two different flows with independent geometric inter-arrival times. Therefore, it is no longer a simple Bernoulli process (for example there can be more than one arrival per time slot). Still, the Markov chain is time-homogeneous, irreducible, finite, and aperiodic and has a simple birth-death structure. The state variable i counts the number of slots in the queue and its steady-state probability distribution can be determined by using the detailed balance equations [19] between neighbouring states:

$$\pi_0 = \frac{1-q}{1-q^{N+1}},\ \forall i : 1 \le i \le N : \pi_i = q^i \pi_0\,,$$

where $q = \frac{r_1 r_2}{(1-r_1)(1-r_2)}$. With the probability of a packet drop at time t due to a full buffer denoted as $D(t)$, the steady-state drop probability can be calculated as

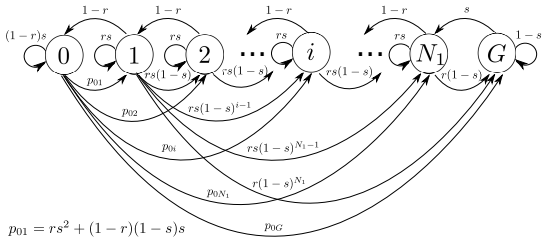
$$p_D = \lim_{t\to\infty} \mathbb{P}(D(t)) = \pi_N\,. \tag{1}$$

The steady-state probability of overload is calculated as

$$p_{O_1} = \sum_{i=N_1+1}^{N} \pi_i = q^{N_1+1} \frac{1-q^{N-N_1}}{1-q^{N+1}}\,.$$

3) Markov Chain for Uncertain Slot Dynamics: The second Markov chain models the position of the first uncertain slot reserved by Class 1 in the buffer spaces numbered 1 to N_1. If an uncertain slot reaches the head of the buffer and no packet of Class 1 is in the packet queue, this slot expires; this is accounted for by State 0 and thus the interest is in the steady-state probability of State 0, denoted as π_0. Another peculiarity of this Markov chain is the State G, which accounts for the situation that there is no uncertain slot in the buffer spaces from 1 to N_1. Remember that the Markov chain is conditioned on the system being in overload. Therefore, from State G new uncertain slots are generated with rate s. This is where the two Markov chains interact and the slot generation rate is set to $s = r_1(1-p_D)$, with p_D calculated from the Markov chain for the total queue length (see Eq. 1). For ease of presentation, s is used to denote the slot generation rate and $r = r_1$ denotes the packet arrival rate of Class 1.

The Markov chain state diagram is shown in Figure 2. While this Markov chain is also time-homogeneous, irreducible, finite, and aperiodic, it is no longer reversible and thus not amenable to detailed balance equations. Instead, the



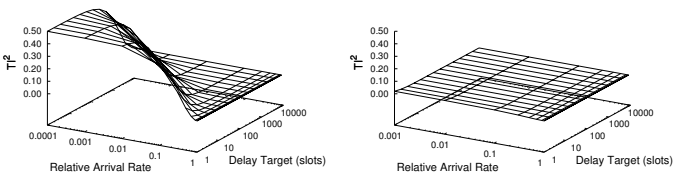
$p_{01} = rs^2 + (1-r)(1-s)s$

$p_{02} = (1-s)(2rs^2 + (1-r)(1-s)s)$

$p_{0i} = (1-s)^{i-1}(irs^2 + (1-r)(1-s)s)$

$p_{0G} = 1 - r\left(N_1(1-s)^{N_1+1} - (N_1+1)(1-s)^{N_1} - 1\right) + (1-s)^{N_1}(1-r)s$

Fig. 2: Markov Chain for Uncertain Slot Dynamics



(a) DD (b) DD+QS

Fig. 3: Throughput Interference - Analysis

global balance equations have to be solved. The following proposition provides the steady-state probability distribution of the uncertain slot Markov chain(see [17] for the proof).

Proposition 1: The steady-state probability distribution for the uncertain slot Markov chain is given as

$$\pi_0 = \frac{1}{1+\sum_{i=1}^{N_1}\frac{\pi_i}{\pi_0}+\frac{\pi_G}{\pi_0}} = \frac{1}{\frac{r\left(\frac{1-s}{1-r}\right)^{N_1}}{s(r-s)} - \frac{1}{r-s} + r(1-s)^{N_1}},$$

$$\pi_i = \frac{(1-s)^{i-1}}{(1-r)^i}\left(1-s\,(1-r)^i\right)\pi_o \quad , i = 1,\ldots,N_1,$$

$$\pi_G = \frac{1}{s}\left(\frac{1-s}{1-r}\right)^{N_1}\left(1-s\,(1-r)^{N_1+1}\right)\pi_o\,.$$

4) Connecting the Decoupled Chains: Both Markov chains are connected again and, thus, the throughput interference under DD can be assessed. Let $r_1 = r$ again and $s = (1-p_D)\,r_1$, as well as $q = \frac{r_1 r_2}{(1-r_1)(1-r_2)}$, then the (steady-state) probability for an expired slot of Class 1 is

$$p_{E_1} = p_{E_1|O_1}\cdot p_{O_1} \quad = \quad \frac{q^{N_1+1}\frac{1-q^{N-N_1}}{1-q^{N+1}}}{\frac{r_1\left(\frac{1-s}{1-r_1}\right)^{N_1}}{s(r_1-s)} - \frac{1}{r_1-s} + r_1(1-s)^{N_1}}.$$

From this, the TI^2 for DD can be derived as

$$TI^2 \approx 1 - \frac{\left(\frac{r_1-p_{E_1}-p_D}{r_1} + \frac{r_2-p_D}{r_2}\right)^2}{2\left(\left(\frac{r_,1-p_{E_1}-p_D}{r_1}\right)^2 + \left(\frac{r_2-p_D}{r_2}\right)^2\right)}.$$

This predicts a very high TI^2 for DD under low-rate traffic with low delay requirements, as shown in Figure 3a for a total load of 1, i.e., $r_1 + r_2 = 1$. In an extreme case, the TI^2 of DD reaches 0.5, which is the maximum TI^2 possible for two flows. Hence, DD on its own, while guaranteeing delay targets by design, does have high throughput interference in certain scenarios. The solution to this problem is the second key principle of DSF: Queue Segmentation. In particular, in DSF the segments are serviced in LIFO order, which is crucial for a low TI^2, because LIFO automatically approximates recent relative arrival rates over a variety of intervals up to the maximum queue length. It thus closely tracks the favourable properties of ICDS [18].

B. Delay Discard & Queue Segmentation

A Markov chain is constructed for Delay Discard and Queue Segmentation (DD+QS). Segment sizes correspond to the delay targets from the previous section, i.e., $S_1 = N_1$ and $S_2 = N_2 - N_1$. A two-segment construction requires a two-dimensional Markov model to track state. Accordingly, the Markov chain is represented with a state tuple (n_2, n_1), where n_1 is the number of packets in the first segment and n_2 is the number of packets in the second segment. Up to a queue size of S_1 the system behaves like FIFO. Once the first segment is full, the behaviour deviates from FIFO, according to the Queue Segmentation principle using LIFO between segments. The next event after service depends on the order of arrival: if a Class 2 packet arrives first, it would create a new slot in the first segment and the Class 1 packet would be discarded without creating any slot; if a Class 1 packet comes first, it would create a slot in the first segment and the Class 2 packet would obtain a slot in the second segment. Assuming that both cases happen with equal probability, the probability of creating a slot for the Class 2 packet in the second segment is $\frac{r_1 r_2}{2}$.

The complete model can be constructed by observing that the second segment size decreases only if the first segment runs empty and no arrival occurs in the same time step ($(1-r_1)\,(1-r_2)$). A sketch of the chain is shown in Figure 4. While it is time-homogeneous, irreducible, finite, and aperiodic, finding an exact closed-form solution is a hard problem, because it is not reversible and the global balance equations are not amenable to a direct solution. To solve the Markov chain for DD+QS, the system is viewed as a queue of queues: each horizontal row is a queue of size S_1 and the vertical dimension is another queue of size S_2. This is an approximation, because the vertical queue's holding times are actually not geometric. The queues are solved independently by starting with the vertical queue and assigning the resulting probability mass to the horizontal queues. The vertical queue system represents each row j as a single state Q_j. Then, each state Q_j has an outgoing edge to Q_{j+1} with probability $(r_1 r_2)/2$ and another one to Q_{j-1} with probability $(1-r_1)\,(1-r_2)$ (except for Q_0 and Q_{S_2}). A local solution for the steady-state distribution of the vertical queue is given by (using $q = \frac{r_1 r_2}{(1-r_1)(1-r_2)}$), for $0 < j \leq S_2$:

$$\pi_{Q_j} = (q/2)^j\ \pi_{Q_0},\ \pi_{Q_0} = \frac{1-q/2}{1-(q/2)^{S_2+1}}\,. \qquad (2)$$

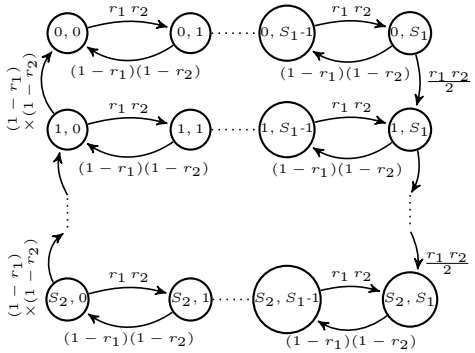


Fig. 4: DD+QS Markov Chain

In order to solve each horizontal queue from Figure 4, observe that each row j has the structure of a simple queue when considering only the transitions between $(Q_j, 0), \ldots, (Q_j, S_1)$ (ignoring transitions between rows). Using states (i) with $i \in 1, \ldots, S_1$ to denote the length of this simple queue, the local solution is obtained, for $0 < i \leq S_1$:

$$\tilde{\pi}_i = q^i\,\tilde{\pi}_0,\ \tilde{\pi}_0 = \frac{1-q}{1-q^{S_1+1}}\,. \tag{3}$$

In order to bring both local solutions together, each row j is normalized to match π_{Q_j}, i.e., $\sum_{i=0}^{S_1} \tilde{\pi}_i = \pi_{Q_j}$. This leads to an approximation for the steady-state distribution of the overall Markov chain for each state (j, i): $\hat{\pi}_{j,i} = \tilde{\pi}_i / \pi_{Q_j}$. This is finally used to obtain the approximate TI^2 for DD+QS.

Proposition 2: A DD+QS scheduler has

$$TI^2 \approx 1 - \frac{\left(1 - \frac{r_2}{2}\sum_{j=0}^{S_2}\hat{\pi}_{j,S_1} + 1 - \frac{r_1}{2}\hat{\pi}_{S_2,S_1}\right)^2}{2\left(\left(1 - \frac{r_2}{2}\sum_{j=0}^{S_2}\hat{\pi}_{j,S_1}\right)^2 + \left(1 - \frac{r_1}{2}\hat{\pi}_{S_2,S_1}\right)^2\right)}$$

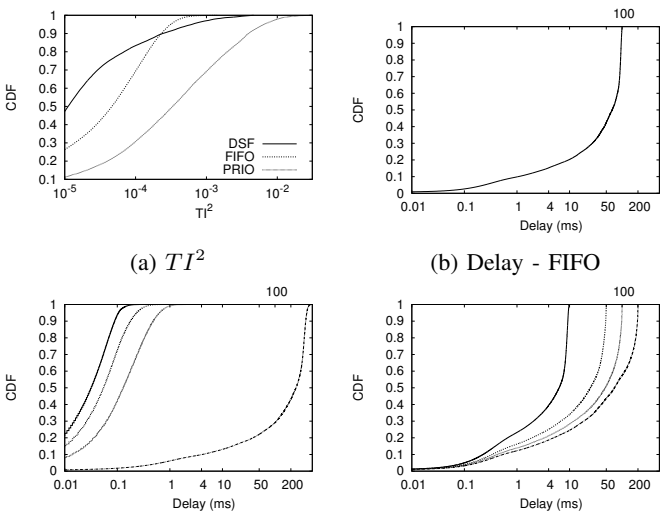
Proof: The steady-state loss rate of each class is derived by combining the local solutions Eq. (2) and Eq. (3). Loss for Class 1 occurs, if there are two arrivals in one of the rightmost states in Figure 4. Therefore, the output rate of Class 1 is $\lambda_1^{out} = r_1 - \frac{r_1 r_2}{2}\sum_{j=0}^{S_2}\hat{\pi}_{j,S_1}$. Loss for Class 2 occurs, if there are two arrivals in state (S_2, S_1). Accordingly, $\lambda_2^{out} = r_2 - \frac{r_1 r_2}{2}\hat{\pi}_{S_2,S_1}$. The final equation then follows according to Definition 1 by letting $\lambda_1^{in} = r_1$ and $\lambda_2^{in} = r_2$. ■

The TI^2 of DD+QS is calculated using Proposition 2 for different delay targets and relative arrival rates and the results are shown in Figure 3b for a total load of 1. Consistently, the TI^2 stays below 0.02, and is clearly much better than for DD only. Furthermore, it is only slightly higher than FIFO's TI^2 (not shown) under the same circumstances, so it can be considered a fairly low value. Based on these results we conclude that using QS and applying the LIFO service principle between queue segments are key to achieving low throughput interference.

VI. Evaluation

All simulations reported in this section are run in the *ns-3* environment and are available online for reproducibility.[1] The simulation scenario is a single-bottleneck dumbbell topology with synthetic workloads. This choice is deliberate. While a simple structured topology and synthetic workloads are not as "realistic" as other setups, they are better suited to develop a systematic understanding of this novel scheduler. The workload keeps the average load around 100%, because this is the interesting operating regime for the scheduler. The default configuration is 100 Mbit/s bottleneck rate, 1ms link propagation delay, 60s simulated time, and 20 repetitions.

[1] https://cs.uwaterloo.ca/%7emkarsten/nidd.html



(a) TI^2 (b) Delay - FIFO

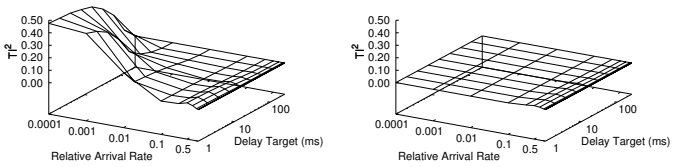
(c) Delay - PRIO (d) Delay - DSF

Fig. 5: TI^2 and Delay Distributions

A. Validation

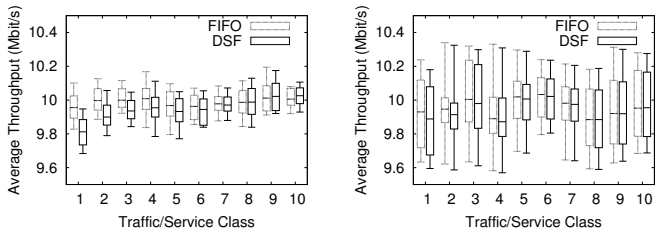
The first experiment is designed to validate the TI^2 and delay characteristics of DSF in comparison with FIFO and static priority scheduling (PRIO). The workload is comprised of 4 traffic classes that each send at 25% of the bottleneck capacity on average. Each traffic class uses 32 on-off traffic sources with on and off periods drawn from a Pareto distribution with shape 1.4. The FIFO and PRIO schedulers are configured with a static buffer equivalent to 100ms queueing delay. With PRIO scheduling, each of the traffic classes uses a separate priority level. The DSF configuration provides 4 delay classes of 10ms, 50ms, 100ms, and 200ms. Each of the traffic classes uses one of the delay classes. The observed values are the average queueing delay and throughput over 10ms time intervals. The TI^2 is computed in 10ms steps using a sliding window of length 1s. The experiment is run once for 600s simulation time, because the observation is longitudinal over time. This longer time interval is sufficient to capture typical random effects. X-axes are shown in logarithmic scale to ensure visibility of all data points.

Figure 5a shows the empirical cumulative distribution function (CDF) of TI^2 values for the different schedulers. It can be seen that PRIO scheduling has worse TI^2 values overall, which corresponds to the understanding that its service rates do not match arrival rates as closely as FIFO or DSF. Note



(a) VIFQ (b) DSF

Fig. 6: Throughput Interference - Packet Simulation



(a) Poisson Traffic (b) Pareto Traffic

Fig. 7: Multiple Classes - FIFO vs. DSF

that the maximum TI^2 of PRIO is recorded as approximately 0.031, which corresponds to a throughput swing of 36%. The equivalent maximum numbers are 2% swing for FIFO and 4% swing for DSF. This observation is complemented by the empirical CDFs for queueing delay shown for each scheduler in Figures 5b-5d. For FIFO's shared queue, there is only one delay curve and average delays go up to the maximum queue length of 100ms. With PRIO, given the load applied in this experiment, the three higher-priority classes show extremely low average delay, which comes at the expense of a high delay of up to 500ms for the lowest-priority class. DSF differentiates the delay for each service class. While these graphs show the average delays over small periods of time, the observed worst-case delays for DSF are in fact bounded by the given targets.

A second experiment revisits the TI^2 comparison between DSF and a system performing only Delay Discard, such as VIFQ. The different TI^2 values predicted by the analytical model in Section V become manifest in comparable packet-level simulations. The simulations are configured to mimic the analytical setup and run for 600s. The results presented in Figure 6 show the long-term TI^2 average over the whole duration of the experiment and thus confirm the structural advantage of adding Queue Segmentation to the scheduler.

B. Traffic Scenarios

The following experiment verifies that DSF is effective and can provide several delay classes – without deviating much from FIFO service in per-class throughput. The DSF scheduler at the bottleneck link is configured with 10 delay classes at 10ms, 20ms, ..., 100ms. Correspondingly, there are 10 traffic classes with an average respective load of 10.1% of the bottleneck capacity; each uses one delay class. In the first part, each traffic class is comprised of 8 Poisson sources.

TABLE II: Traffic Mix - Baseline for 100 Mbit/s

Traffic Class	Source Type (RTT, if apl)	Flows	Load or File Size	Delay Class	Notes
1	Poisson	160	64Kbit/s	5ms	*Voice*
2	Pareto 1.6	160	192Kbit/s	100ms	*Video*
3a	TCP (20ms)	20	10KB	(*)	*Data*
3b		10	100KB	(*)	
3c		5	1000KB	(*)	
3d		5	infinite	(*)	
4a	TCP (200ms)	20	10KB	200ms	*Data*
4b		10	100KB	200ms	
4c		5	1000KB	200ms	
4d		5	infinite	200ms	
5a	Pareto 1.4	16	150Kbit/s	10ms	*Other*
5b		16	150Kbit/s	20ms	
5c		16	150Kbit/s	100ms	
5d		16	150Kbit/s	200ms	

(*) delay class 20ms or 200ms

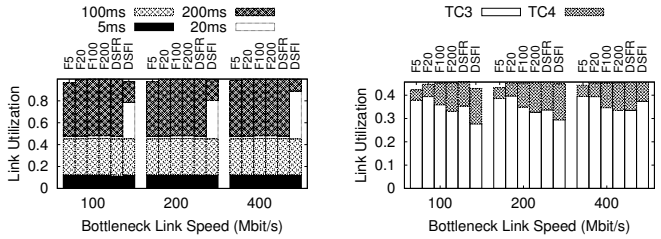
In a second part, each traffic class is comprised of 32 Pareto sources with a shape value of 1.4. The DSF configuration is compared to a FIFO system with 100ms buffer.

Figure 7 shows the distribution of average throughput for each traffic/service class. The box-and-whisker plots show the average throughput, as well as the 10%, 25%, 75%, and 90% quantiles. The results also confirm the conjectured properties of DSF. The average throughput per class almost perfectly matches FIFO throughput. Traffic sources loose a slight but increasing fraction of packets when choosing smaller delay targets – evident when comparing the mean values for FIFO and DSF in Figure 7 across service classes. The experiment has been repeated with various numbers around 100% total average load without any significant change in outcome.

C. TCP

DSF provides delay guarantees for any number of service classes with very low throughput interference. However, TCP has two effects that relate buffering to goodput: 1) Higher levels of flow multiplexing reduce the overall amount of buffering needed at the bottleneck to achieve good link utilization [3], [29]. 2) Long-term TCP goodput is inversely proportional to the RTT [23]. Thus, DSF potentially has indirect side effects on TCP goodput through both delay control and buffer management. This is investigated by testing a comprehensive traffic mix with FIFO configurations for 5ms, 20ms, 100ms, and 200ms, as well as two DSF scenarios with 4 service classes for the four delay targets. The *regular* (DSFR) scenario assumes that all TCP traffic uses the default 200ms delay class, regardless of propagation delay. TCP traffic is typically throughput-oriented and at 200ms, long-term TCP throughput is not affected by bufferbloat [10] effects. In contrast, the *impatient* (DSFI) scenario investigates the effect of TCP traffic with smaller RTTs using a smaller delay class. Based on the classical notion of RTT unfairness and the concerns stated in the ABE [14] and RD [25] proposals, this could be expected to result in a highly imbalanced bottleneck capacity allocation.

A synthetic traffic mix, specified in Table II, is loosely modelled after Sandvine's Global Internet Phenomena Report [27],



(a) Throughput (b) TCP Goodput

Fig. 8: Traffic Mix

which reports traffic comprised of real-time entertainment (40-60%), web browsing and social networking (15-30%), and communication services (5-10%). *Data* traffic uses TCP NewReno while *Voice*, *Video*, and *Other* traffic is modelled as UDP. Each *Data* traffic flow performs renewing file transfers with sizes drawn from a Pareto distribution with shape 1.2. The traffic mix is scaled to various bottleneck link rates by proportionally scaling the number of flows in each traffic class.

Figure 8a shows the link-level throughput measurements for each service class at different bottleneck link rates. It confirms that DSF does not unduly influence throughput of service classes. DSFI shows a higher throughput for the 20ms classes, because the TCP flows from TC3 enter this class. To properly investigate TCP-level effects, Figure 8b shows the aggregate TCP goodput for each of the TCP traffic classes. The general problem with a small-buffer configuration is visible for F5 and F20, albeit not as strong as prior work has shown in certain scenarios (cf. Section III in [17]). The nature of TCP's RTT unfairness is visible throughout all configurations, because TC3 generally obtains a significantly higher share of the bottleneck capacity than TC4. When taking F100 or F200 as the benchmark, DSFR performs equally well in terms of goodput and RTT unfairness, and much better than F5 or F20. This demonstrates the essential benefits of DSF. The results for DSFI are somewhat inconclusive. In the 100 Mbit/s experiment, RTT unfairness is reduced, while overall utilization suffers. In the 200 Mbit/s configuration, DSFI outperforms all other scheduler configurations. At 400 Mbit/s link speed, RTT unfairness is slightly higher, but DSFI is still better than F5 and F20. Thus, even in scenarios where the classical TCP models predict RTT unfairness, DSF appears competitive.

VII. Conclusion

The goal of this work is providing multiple delay classes while adhering to strict network neutrality requirements. The DSF scheduler is presented as an effective solution. DSF can provide multiple service classes with arbitrary delay targets. It is minimal and modular by focusing on a single feature (delay differentiation) with very little side effects. It is shown analytically and with simulations that delays are enforced without significant throughput interference. A non-intuitive outcome is that DSF does not necessarily cause additional RTT unfairness for TCP traffic. This poses an interesting challenge to possibly refine existing TCP models.

References

[1] 3GPP Specification detail - 3GPP TS 23.203 Policy and charging control architecture. http://www.3gpp.org/DynaReport/23203.htm.

[2] S. Aalto, U. Ayesta, S. Borst, V. Misra, and R. Núñez Queija. Beyond Processor Sharing. *ACM Perform. Eval. Rev.*, 34(4):36–43, Mar. 2007.

[3] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *ACM SIGCOMM*, pages 281–292. ACM, 2004.

[4] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475 - An Architecture for Differentiated Services, Dec. 1998.

[5] S. Cheung and C. Pencea. Pipelined Sections: A New Buffer Management Discipline for Scalable QoS Provision. In *IEEE INFOCOM*, pages 1530–1538, Apr. 2001.

[6] J. P. Choi, D.-S. Jeon, and B.-C. Kim. Net Neutrality, Business Models, and Internet Interconnection. *AEJ Microeconomics*, 2014.

[7] J. Crowcroft. Net Neutrality: The Technical Side of the Debate: A White Paper. *ACM Comput. Commun. Rev.*, 37(1):49–56, 2007.

[8] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: Re-architecting Congestion Control for Consistent High Performance. In *USENIX NSDI*, pages 395–408, May 2015.

[9] FCC. 15-24: Protecting and Promoting the Open Internet, March 2015.

[10] J. Gettys and K. Nichols. Bufferbloat: Dark Buffers in the Internet. *Commun. ACM*, 55(1):57–65, Jan. 2012.

[11] A. Gravey, J.-R. Louvion, and P. Boyer. On the Geo/D/1 and Geo/D/1/n Queues. *Perform. Eval.*, 11(2):117–125, 1990.

[12] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues Don't Matter When You Can JUMP Them! In *USENIX NSDI*, 2015.

[13] R. Guerin, S. Kamat, V. Peris, and R. Rajan. Scalable QoS Provision Through Buffer Management. In *ACM SIGCOMM*, Aug. 1998.

[14] P. Hurley, J.-Y. L. Boudec, P. Thiran, and M. Kara. ABE: Providing a Low-Delay Service within Best-Effort. *IEEE Network*, 15(3):60–69, May 2001.

[15] R. Jain, D.-M. Chiu, and W. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. Technical report, Sept. 1984. DEC Research Report TR-301.

[16] M. Karsten. FIFO Service with Differentiated Queueing. In *ACM/IEEE ANCS*, pages 122–133, Oct. 2011.

[17] M. Karsten, D. S. Berger, and J. Schmitt. Traffic-Driven Implicit Buffer Management (ext.), 2016. http://cs.uwaterloo.ca/%7emkarsten/dsftr.pdf.

[18] M. Karsten, K. Larson, and Y. Lin. Incentive-Compatible Differentiated Scheduling. In *ACM HotNets*, pages 101–106, Nov. 2005.

[19] F. P. Kelly. *Reversibility and Stochastic Networks*. Wiley, 1979.

[20] N. Kuhn, E. Lochin, and O. Mehani. Revisiting Old Friends: Is CoDel Really Achieving What RED Cannot? In *ACM SIGCOMM Capacity Sharing Workshop*, pages 3–8, 2014.

[21] M. A. Lemley and L. Lessig. End of End-to-End: Preserving the Architecture of the Internet in the Broadband Era. *UCLA Law Review*, 48:925, 2000.

[22] M. Mathis. Reflections on the TCP Macroscopic Model. *ACM Comput. Commun. Rev.*, 39(1):47–49, Dec. 2008.

[23] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Comput. Commun. Rev.*, 27(3):67–82, July 1997.

[24] K. Nichols and V. Jacobson. Controlling Queue Delay. *ACM Queue*, 10(5):20–34, May 2012.

[25] M. Podlesny and S. Gorinsky. RD Network Services: Differentiation through Performance Incentives. In *ACM SIGCOMM*, pages 255–266, 2008.

[26] R. Ravaioli, G. Urvoy-Keller, and C. Barakat. Towards a General Solution for Detecting Traffic Differentiation at the Internet Access. In *International Teletraffic Congress*, pages 1–9, Sept. 2015.

[27] SANDVINE. Global Internet Phenomena Report: 1H 2015, May 2015.

[28] B. van Schewick. Analysis of Proposed Network Neutrality Rules. *Stanford Center for Internet and Society*. February 18, 2015.

[29] A. Vishwanath, V. Sivaraman, and M. Thottan. Perspectives on Router Buffer Sizing: Recent Results and Open Problems. *ACM Comput. Commun. Rev.*, 39(2):34–39, Mar. 2009.

[30] T. Wu and C. Yoo. Keeping the Internet Neutral? *Federal Communications Law Journal*, 59:575–615, 2007.

[31] C. S. Yoo. Beyond Network Neutrality. *Harvard Journal of Law and Technology*, 19, 2005.

[32] K. Zhu. Bringing Neutrality to Network Neutrality. *Berkeley Technology Law Journal*, 22:615, 2007.