Traffic-Driven Implicit Buffer Management (Extended Version)

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 [54].

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 [18].
- 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 [18], [39], [57], [63]. It has evolved from the question whether cable modem Internet providers in the USA should be regulated under the same "common carrier" rules as providers using phone lines [4], [40], [41], [55], [56], [64], [65]. In this context, Lemley and Mark [39] use the notion of the Internet's "end-to-end architecture" to derive a "principle of non-discrimination among applications". This idea has been refined into the term "neutral network – that is, an

Internet that does not favour one application [..] over others" by Wu [60]. According to Wu [62], a neutral network is essential for innovation on the Internet, which he observes to happen mainly at the network edge. Consequently, Wu proposes rules to enforce a neutral network in the future [60], [61], [62].

Other legal scholars have contested the necessity of such rules, e.g., by reasoning that competition forces Internet service providers to maximize their value to customers and thus allow for a variety of applications [55], [56], [64]. For example, Yoo [65] uses delay demands of voice communication as an example to highlight that not all applications are served well by a network with equal treatment of all packets. Neutrality rules would prohibit some solution approaches, such as priority service for delay-sensitive applications, and could thus also stifle innovation.

From a technical perspective, Crowcroft [12] criticizes the discussion at the time as largely oversimplified. He argues that the Internet has never been "innately fair (for whatever definition of fairness you wish to choose [..])" and he concludes that the Internet was never network neutral. To back this claim, he names various examples of inherent biases of the Internet architecture such as the dependence of TCP throughput on round trip times, and common practices, such as admission control for caching nodes, or service level agreements on statistical performance guarantees.

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 [11], [65], [68].

While the academic discussion on legal and economic aspects of network neutrality is still ongoing, several countries have considered various approaches for regulation. In the USA, a consensus has evolved to "ban paid prioritization, throttling, and blocking" [58], represented by the recent FCC Open Internet Order [18].

The literature about detecting network neutrality violations typically uses a broader definition. There is a large body of work in this area [5], [14], [15], [33], [34], [43], [46], [66], [67] with many claims of finding evidence for various degrees of network neutrality violations. In recent work [51], 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 [32] 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 Appendix A. Secondly, this definition incorporates the no blocking and no throttling rules [18] because it ensures that relative service rates correspond to relative arrival rates. Thirdly, this definition incorporates the loss-based metric from recent work [51], 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 [18]. 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 [36] 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 [31] 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) [35] is a previous attempt to generalize ABE to multiple classes and to implement ICDS [36]. 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 [19], [20]. The most recent incarnations of this approach are the RD proposal [50] and QJump [25]. 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 [45]. 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. In both ABE and RD, addressing TCP's "RTT unfairness" thus requires a complex and brittle parameter estimation.

Mathis [44] has questioned this traditional, narrow focus on TCP-friendly rate control. For example, recent alternative TCP rate control algorithms [16] 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.

Traditional QoS schedulers that fit into the context of the IETF's IntServ [7] or DiffServ [6] architectures are either based on exclusive resource allocation [9], [49] or priorities [17], [42]. In both cases, admission control is needed to avoid the *tragedy of the commons* effect [27]. Thus, these systems only work, if access to service classes is restricted. The same is true for service differentiation through congestion pricing [28], [48], [52], which degenerates into paid prioritization under congestion.

Several implementation proposals use differentiated buffer management to differentiate service [10], [26]. 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.

Last but not least, a straightforward approach to delay control is limiting the maximum queueing delay of every packet by a using small shared buffer. Because the traffic aggregate of all applications is jointly controlled, the application with the smallest delay target dictates the buffer size. This idea has received extensive research interest since an earlier proposal for small router buffers [3]. Smaller router buffers, however, sacrifice link utilization and increase packet loss, and are thus only applicable in some scenarios [8], [13].

Another approach has recently (re)gained research interest: controlling queueing delay through active queue management (AQM). A prominent example is the CoDel algorithm [47]. 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

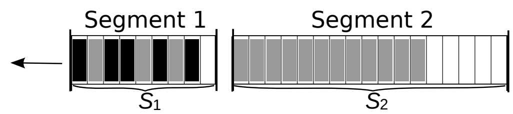


Fig. 1: Queue Segmentation

delay and high goodput, and indeed, a recent evaluation shows that CoDel achieves the best goodput among several other AQMs [38]. Nevertheless, CoDel does not solve the fundamental trade-off between delay and goodput [30]. In some scenarios, CoDel can increase the file completion time by 42% [38]. In summary, there is no conclusive evidence that small router buffers or AQM can support all desirable delay targets under all circumstances without negative side effects . This is illustrated in Appendix B with a set of simulation experiments.

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 [6]. 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 [35] 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.

Therefore, DSF introduces *Queue Segmentation* in the buffer organization: each delay target is represented by a corresponding queue segment. This is illustrated by a simple two-class example. Figure 1 shows a buffer that is split into two segments of size S_1 and S_2, with slots from two service classes in black and grey. S_1 corresponds to the lower delay target, while $S_1 + S_2$ corresponds to the higher delay target. The scheduler places all new slots (from both classes) into Segment 1 – as long as there is space. Slots in Segment 1 are served in the normal FIFO order, which creates new room in this segment. If at the time of packet arrival, Segment 1 is full, only slots for the higher delay class are accepted into Segment 2 – as long as there is space. However, slots from Segment 2 are

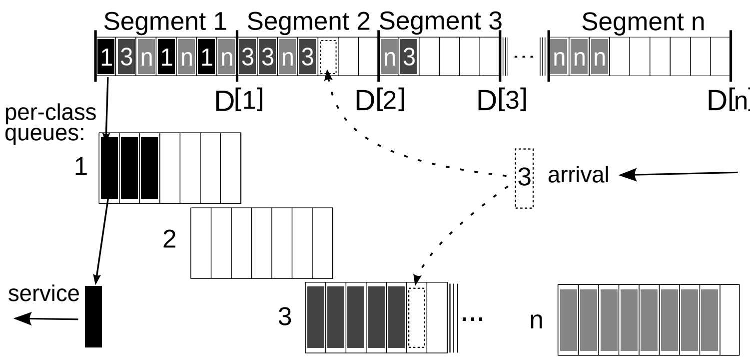


Fig. 2: Delay Segment FIFO

served only when Segment 1 runs empty. In DSF, this scheme is used for a number of classes. 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. The rationale for this change is further explained in Section V. 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. The combination of Delay Discard and Queue Segmentation gives DSF its name.

A. *Algorithm*

DSF is illustrated in Figure 2 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 (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

Algorithm 1 DSF Arrival Routine (per packet)

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

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

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.

Algorithm 2 DSF Service Routine (loop)

```
i ← 1
while squeue[i].empty() do i ← i + 1
{c, s} ← squeue[i].pop()
buffer[c] ← buffer[c] - s
service[c] ← service[c] + s
while service[c] ≥ 0 and pqueue[c].nonempty() do
    {p, d} ← pqueue[c].pop()
    if now() < d then
        service[c] ← service[c] - size(p)
        send(p)
```

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.

The LIFO service principle is applied across slot segments, while packets are stored in and transmitted from the per-class packet queues. This decoupling of slots and packets permits reordering service times between slots, while maintaining the ordering of packets within a class. An interesting observation is that the DSF system degenerates to per-packet LIFO service, if delays are so close to each other that each segment can hold only a single slot. However, considering contemporary line rates and meaningful delay targets, this is unlikely to happen in practice.

During busy periods, the system temporarily operates like a short FIFO queue corresponding to the smallest delay target. Excess arrivals to this short queue are not immediately discarded though, but corresponding slots are stored in higher-order slot segments, as much as possible. These slots can be used during underload queue drain periods to transmit packets that would not have been accepted into the short FIFO queue. Service classes with a higher delay target have access to more slot segments. Overall, DSF does not treat packets of any service class inherently better or worse than any other service class. The only service differentiation taking place is enforcing the self-imposed delay targets chosen by end systems. The main question is whether DSF provides service rates in proportion to arrival rates to achieve a low TI^2 value. This is further investigated in Sections V and VI.

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. An asynchronous cleanup procedure can remove old packets and turn the test in Line 8 into a pure safety measure, thus avoiding looping even more. The current DSF algorithm is only a starting point and we expect that improved versions can be found.

V. Analysis

This section presents analytical models for the two key principles of DSF: Delay Discard and Queue Segmentation. We provide abstract discrete-time Markov chain models for these in order to find closed form expressions for the TI^2. For the ease of presentation and to keep the models analytically tractable we make a set of simplifying assumptions: The models 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. The size of the state space is given in Appendix C.

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 [23] 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 [37] between neighbouring states:

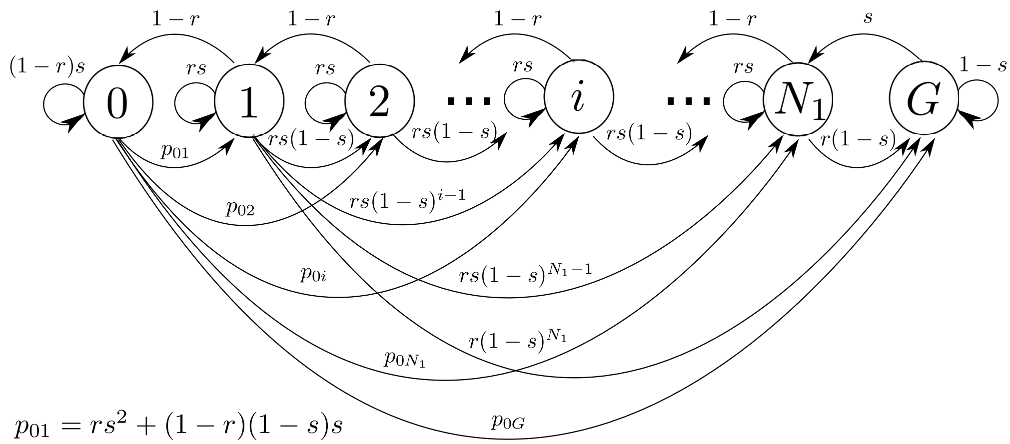
$$\pi_0 = \frac{1-q}{1-q^{N+1}}, \ \forall i : 1 \le i \le N : \pi_i = q^i \pi_0 \,,$$

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

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



$$
\begin{aligned}
p_{01} &= rs^2 + (1-r)(1-s)s \\
p_{02} &= (1-s)(2rs^2 + (1-r)(1-s)s) \\
p_{0i} &= (1-s)^{i-1}(irs^2 + (1-r)(1-s)s) \\
p_{0G} &= 1 - r\left(N_1(1-s)^{N_1+1} - (N_1+1)(1-s)^{N_1} - 1\right) + (1-s)^{N_1}(1-r)s
\end{aligned}
$$

Fig. 3: Markov Chain for Uncertain Slot Dynamics

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 3. 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 global balance equations have to be solved. The following proposition provides the steady-state probability distribution of the uncertain slot Markov chain.

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

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

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

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

Proof: The global balance equations are used for an induction over state $i = 1, \ldots, N_1$. The induction hypothesis is the statement of the proposition regarding the $\pi_i, i = 1, \ldots, N_1$:

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

Basis: We start with the global balance equation for state 0

$$1 - (1-r)s\pi_0 = (1-r)\pi_1 \Rightarrow \pi_1 = \frac{1 - s(1-r)}{1-r}\pi_0 = \frac{(1-s)^0}{(1-r)^1}\left(1 - s(1-r)^1\right)$$

which provides the induction hypothesis for $i = 1$.

Inductive step: Now we use the global balance equation for state $i > 1$, in particular:

$$(1-r)\pi_{i+1} + \sum_{j=1}^{i-1} rs(1-s)^{i-j}\pi_j + \left(irs^2(1-s)^{i-1} + (1-r)s(1-s)^i\right)\pi_0 = (1-rs)\pi_i \tag{2}$$

Using the induction hypothesis for all π_j with $j \leq i$ and some rearranging we obtain

$$\begin{aligned}
\pi_{i+1} = \frac{\pi_0}{1-r}\Bigg[&(1-rs)\frac{(1-s)^{i-1}}{(1-r)^i}\left(1 - s(1-r)^i\right) - \sum_{j=1}^{i-1} rs(1-s)^{i-j}\frac{(1-s)^{j-1}}{(1-r)^j}\left(1 - s(1-r)^j\right) \\
&- \left(irs^2(1-s)^{i-1} + (1-r)s(1-s)^i\right)\Bigg] \\
=&\frac{(1-s)^{i-1}}{(1-r)^{i+1}}\pi_0\Bigg[(1-rs)\left(1 - s(1-r)^i\right) - \sum_{j=1}^{i-1} rs(1-r)^{i-j}\left(1 - s(1-r)^j\right) \\
&- (1-r)^i irs^2 - (1-r)^{i+1}s(1-s)\Bigg] \\
=&\frac{(1-s)^{i-1}}{(1-r)^{i+1}}\pi_0\Bigg[1 - s(1-r)^i - \sum_{j=1}^{i} rs(1-r)^{i-j}\left(1 - s(1-r)^j\right) \\
&- (1-r)^i irs^2 - (1-r)^{i+1}s(1-s)\Bigg] \\
=&\frac{(1-s)^{i-1}}{(1-r)^{i+1}}\Bigg[\pi_0 1 - s(1-r)^i - \sum_{j=1}^{i} rs(1-r)^{i-j} - \sum_{j=1}^{i} rs^2(1-r)^{i-j} \\
&- (1-r)^i irs^2 - (1-r)^{i+1}s(1-s)\Bigg] \\
=&\frac{(1-s)^{i-1}}{(1-r)^{i+1}}\pi_0\left[1 - s(1-r)^i - \sum_{j=1}^{i} rs(1-r)^{i-j} - (1-r)^{i+1}s(1-s)\right] \\
=&\frac{(1-s)^{i-1}}{(1-r)^{i+1}}\pi_0\left[1 - s(1-r)^i - s\left(1 - (1-r)^i\right) - (1-r)^{i+1}s(1-s)\right] \\
=&\frac{(1-s)^{i}}{(1-r)^{i+1}}\left(1 - s(1-r)^{i+1}\right)\pi_0\,,
\end{aligned}$$

thereby showing that the rule in the proposition indeed holds for π_{i+1}.

π_G has to be solved separately using the global balance equation for state N_1, in particular

$$s\pi_g + \sum_{j=1}^{N_1-1} rs(1-s)^{N_1-j}\pi_j + \left(N_1 rs^2(1-s)^{N_1-1} + (1-r)s(1-s)^{N_1}\right)\pi_0 = (1-rs)\pi_{N_1}.$$

This has the same form as Eq. 2, apart from the factor s in the first term (instead of $1-r$). Thus, essentially the same derivation as in the inductive step above can be performed to obtain

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

π_0 is obtained from the normalization condition as

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

Note that in the third line we assumed $s \neq r$, which however is always true for a finite queue (in fact, we have $s < r$). This concludes the proof. ■

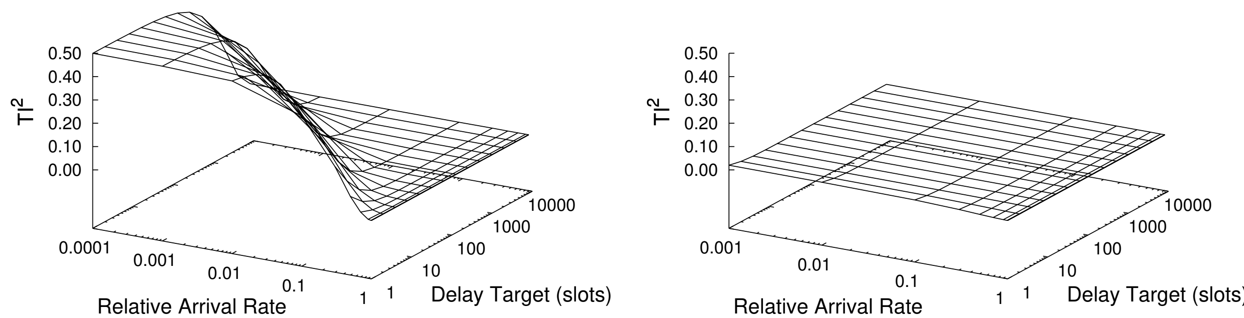
4) Connecting the Decoupled Chains: Both Markov chains are connected again and, thus, the throughput interference under DD can be assessed. Let $r_1 = r$ again and $s = (1-p_D)\,r_1$, as well as $q = \frac{r_1 r_2}{(1-r_1)(1-r_2)}$, then the (steady-state) probability for an expired slot of Class 1 is

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

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 4a 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 [36].



(a) DD

(b) DD+QS

Fig. 4: Throughput Interference - Analysis

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 5. 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

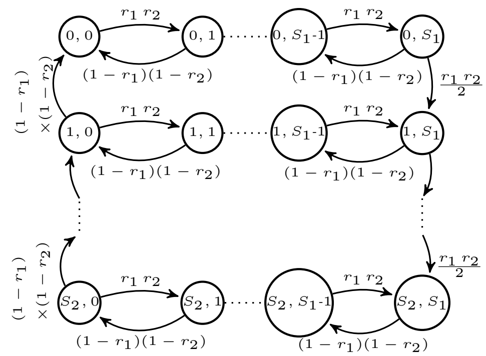


Fig. 5: DD+QS Markov Chain

distribution of the vertical queue is given by (using $q = \frac{r_1 r_2}{(1-r_1)(1-r_2)}$), for $0 < j \leq S_2$:

$$\pi_{Q_j} = (q/2)^j\ \pi_{Q_0},\ \pi_{Q_0} = \frac{1-q/2}{1-(q/2)^{S_2+1}}\ . \tag{3}$$

In order to solve each horizontal queue from Figure 5, 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 $0 < i \leq S_1$:

$$\tilde{\pi}_i = q^i\ \tilde{\pi}_0,\ \tilde{\pi}_0 = \frac{1-q}{1-q^{S_1+1}}\ . \tag{4}$$

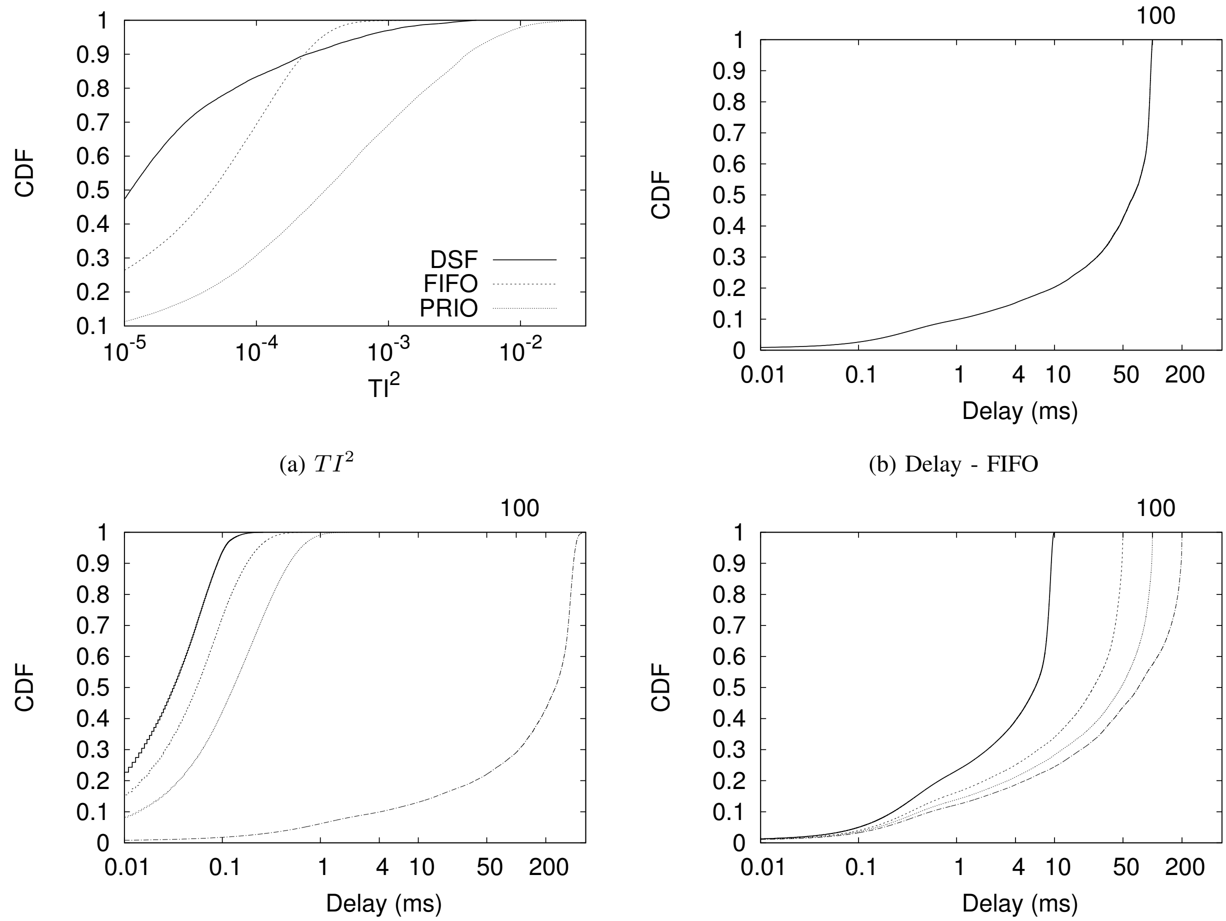
In order to bring both local solutions together, each row j is normalized to match π_{Q_j}, i.e., $\sum_{i=0}^{S_1} \tilde{\pi}_i = \pi_{Q_j}$. This leads to an approximation for the steady-state distribution of the overall Markov chain for each state (j, i): $\hat{\pi}_{j,i} = \tilde{\pi}_i / \pi_{Q_j}$. 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. (3) and Eq. (4). Loss for Class 1 occurs, if there are two arrivals in one of the right-most states in Figure 5. 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 4b 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.



(a) TI^2

(b) Delay - FIFO

(c) Delay - PRIO

(d) Delay - DSF

Fig. 6: TI^2 and Delay Distributions

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/`

A. Validation

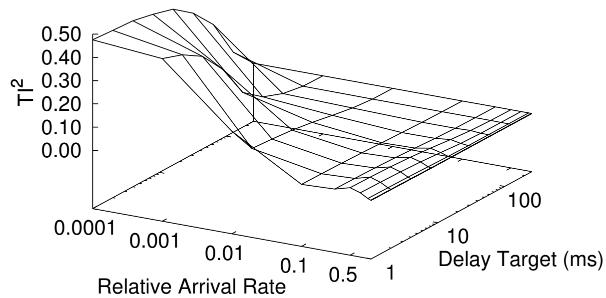
The first experiment is designed to validate the TI^2 and delay characteristics of DSF in comparison with FIFO and static priority scheduling (PRIO). The workload is comprised of 4 traffic classes that each send at 25% of the bottleneck capacity on average. Each traffic class uses 32 on-off traffic sources 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 6a 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. The effects of PRIO scheduling primarily exist over shorter time intervals, which is briefly discussed in Appendix A. Note 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 6b-6d. 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.

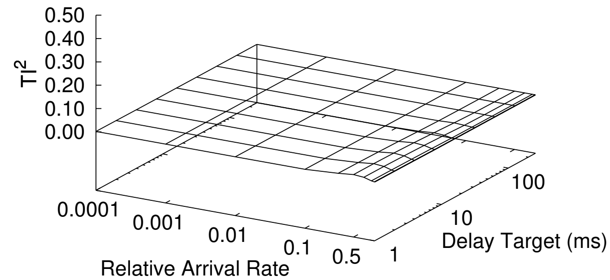
A second experiment revisits the TI^2 comparison between DSF and a system performing only Delay Discard, such as VIFQ. The different TI^2 values predicted by the analytical model in Section V become manifest in comparable packet-level simulations. The simulations are configured to mimic the analytical setup and run for 600s. The results presented in Figure 7 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 next experiment illustrates the basic overall efficacy of DSF in an extreme setup. A single VoIP-type traffic flow of 64 Kbit/s shares the bottleneck with 50 Pareto sources with an aggregate mean rate identical to the bottleneck link capacity. The experiment is run in a FIFO configuration with 100ms buffer space and a DSF configuration with two service classes - 1ms for VoIP, and 100ms for other traffic. The observed values are the average and maximum delay as well as the throughput for each of the traffic classes measured for non-overlapping intervals of 1s. Figure 8



(a) VIFQ



(b) DSF

Fig. 7: Throughput Interference - Packet Simulation

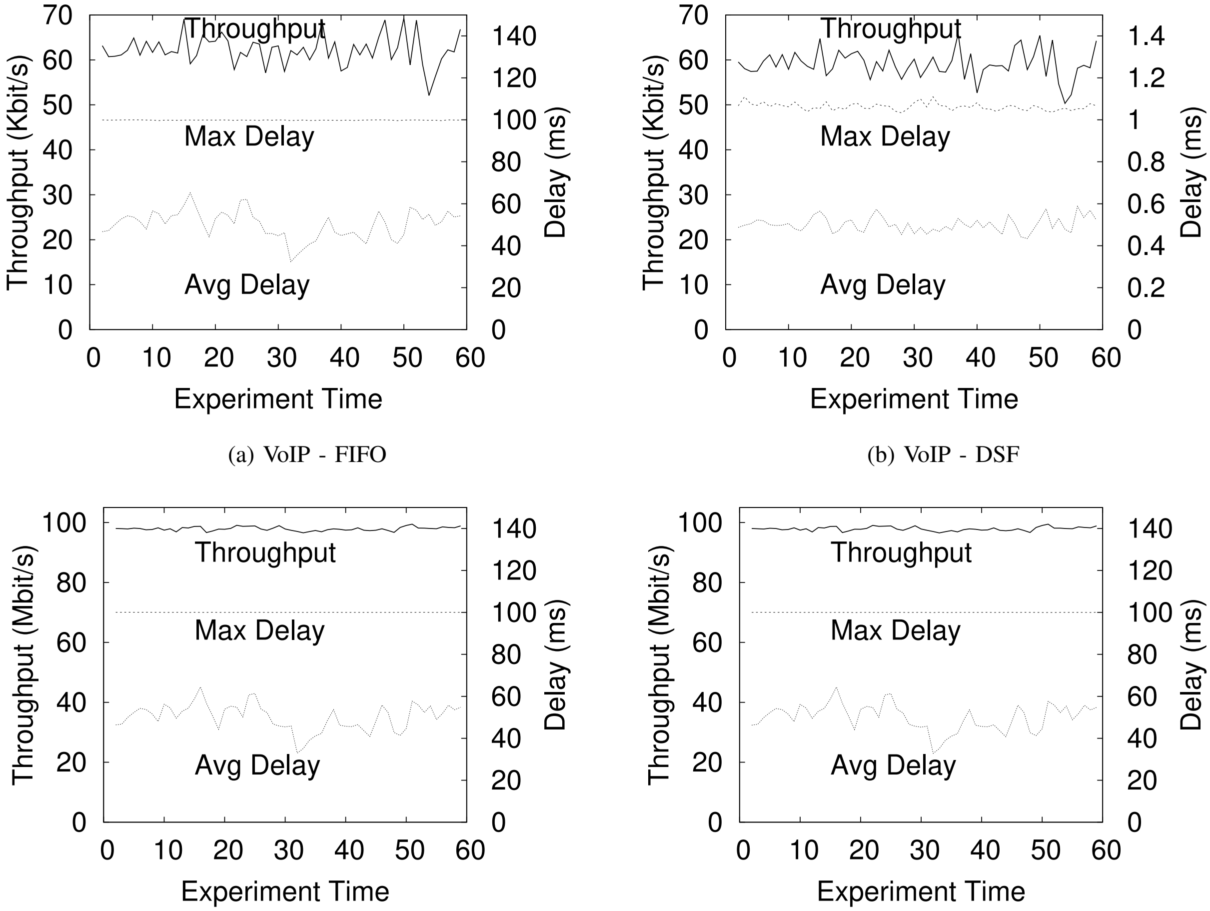
shows the results. As expected, the target delay is met, but DSF also ensures that low-delay traffic has appropriate throughput. The background traffic does not suffer a throughput or delay penalty under DSF.

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. 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 9 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 9 across service classes. The experiment has been repeated with various numbers around 100% total average load, such as 99%, 99.9%, 100%, 100.1% or 105%, 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], [59]. 2) Long-term TCP goodput is inversely proportional to the RTT [45]. 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



(a) VoIP - FIFO

(b) VoIP - DSF

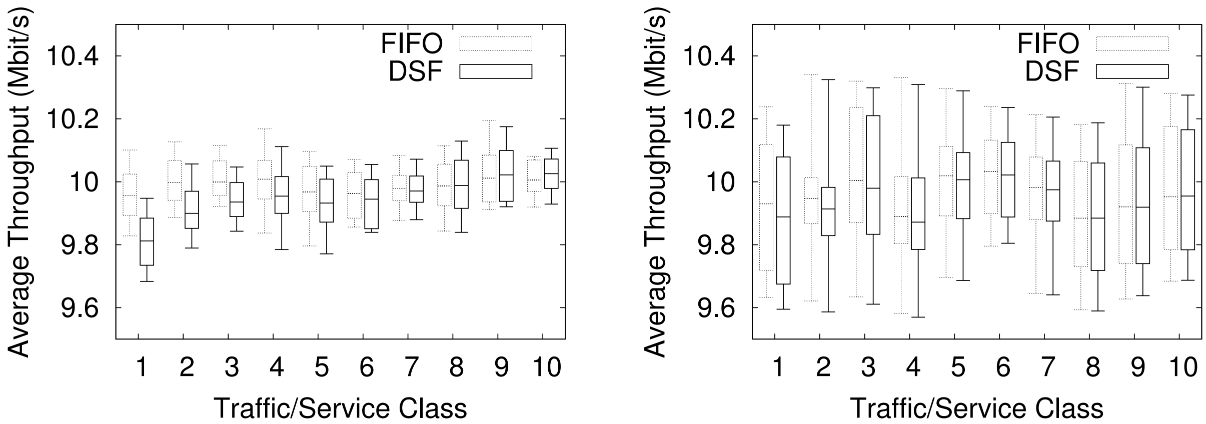
(c) Background - FIFO

(d) Background - DSF

Fig. 8: Single Flow - Note different y2 ranges!

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 [21] 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 [31] and RD [50] 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 [53], 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) Poisson Traffic

(b) Pareto Traffic

Fig. 9: Multiple Classes - FIFO vs. DSF

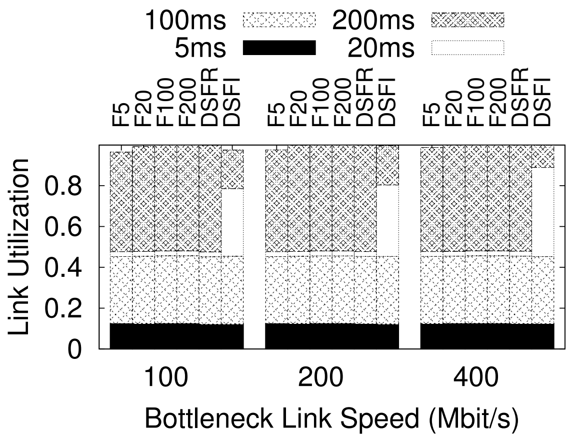
TABLE II: Traffic Mix - Baseline for 100 Mbit/s

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	

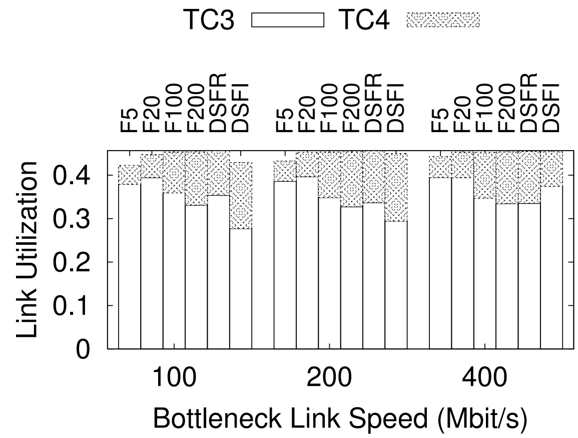
(*) delay class 20ms or 200ms

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 10a 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 10b shows the aggregate TCP goodput for each of the TCP traffic classes. The general problem with



(a) Throughput



(b) TCP Goodput

Fig. 10: Traffic Mix

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). 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] F. Bar, S. Cohen, P. Cowhey, B. DeLong, M. Kleeman, and J. Zysman. Defending the Internet Revolution in the Broadband Era: When Doing Nothing is Doing Harm, August 1999. E-conomy Working Paper 12.

[5] R. Beverly, S. Bauer, and A. Berger. The Internet is Not a Big Truck: Toward Quantifying Network Neutrality. In *ACM PAM*, pages 135–144. Springer, 2007.

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

[7] R. Braden, D. Clark, and S. Shenker. RFC 1633 - Integrated Services in the Internet Architecture: an Overview, June 1994.

[8] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I.-J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl. Reducing Internet Latency: A Survey of Techniques and their Merits. *IEEE Commun. Surveys Tuts.*, 2014. http://dx.doi.org/10.1109/COMST.2014.2375213, to appear.

[9] F. Checconi, L. Rizzo, and P. Valente. QFQ: Efficient Packet Scheduling with Tight Guarantees. *IEEE/ACM Trans. Netw.*, 21(3):802–816, June 2013.

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

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

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

[13] A. Dhamdhere and C. Dovrolis. Open Issues in Router Buffer Sizing. *ACM Comput. Commun. Rev.*, 36(1):87–92, 2006.

[14] M. Dischinger, M. Marcon, S. Guha, K. Gummadi, R. Mahajan, and S. Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *USENIX NSDI*, April 2010.

[15] M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi. Detecting BitTorrent Blocking. In *ACM IMC*, pages 3–8, 2008.

[16] 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.

[17] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. *IEEE/ACM Trans. Netw.*, 10(1):12–26, Feb. 2002.

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

[19] V. Firoiu and X. Zhang. Best Effort Differentiated Services: Trade-off Service Differentiation for Elastic Applications. In *IEEE ICT 2001*, June 2001.

[20] B. Gaidioz and P. Primet. EDS: A New Scalable Service Differentiation Architecture for Internet. In *IEEE ISCC 2002*, pages 777–782, July 2002.

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

[22] Y. Ghiassi-Farrokhfal and J. Liebeherr. Output Characterization of Constant Bit Rate Traffic in FIFO Networks. *IEEE Comm. Letters*, 13(8):618–620, 2009.

[23] 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.

[24] G. Grimmet and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2001.

[25] 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*, pages 1–14, May 2015.

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

[27] G. Hardin. The Tragedy of the Commons. *Science*, 162(3859):1243–1248, 1968.

[28] T. Henderson, J. Crowcroft, and S. Bhatti. Congestion Pricing: Paying Your Way in Communication Networks. *IEEE Internet Computing*, 5(5):85–89, Sept. 2001.

[29] D. P. Heyman and M. J. Sobel. *Stochastic Models in Operations Research*, volume 1. McGraw Hill, 1982.

[30] G. Hill and C. Shah. Reproducing Network Research Blog: CS244'15: CoDel – Controlling Delay in Queues. `https:// reproducingnetworkresearch.wordpress.com/2015/05/31/cs244-15-codel-controlling-delay-in-queues`.

[31] 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.

[32] 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.

[33] P. Kanuparthy and C. Dovrolis. Diffprobe: Detecting ISP Iervice Discrimination. In *IEEE INFOCOM*, pages 1–9, 2010.

[34] P. Kanuparthy and C. Dovrolis. ShaperProbe: End-to-End Detection of ISP Traffic Shaping using Active Methods. In *ACM IMC*, pages 473–482, 2011.

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

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

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

[38] 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.

[39] 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.

[40] L. Lessig. The Cable Debate, Part II. *The Industry Standard*, page 2, January 1999.

[41] L. Lessig. The Internet Under Siege. *Foreign Policy*, 127:56, 2001.

[42] M. K. H. Leung, J. C. S. Lui, and D. K. Y. Yau. Adaptive Proportional Delay Differentiated Services: Characterization and Performance Evaluation. *IEEE/ACM Trans. Netw.*, 9(6):801–817, Dec. 2001.

[43] G. Lu, Y. Chen, S. Birrer, F. E. Bustamante, C. Y. Cheung, and X. Li. End-to-End Inference of Router Packet Forwarding Priority. In *IEEE INFOCOM*, pages 1784–1792, 2007.

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

[45] 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.

[46] A. Molavi Kakhki, A. Razaghpanah, R. Golani, D. Choffnes, P. Gill, and A. Mislove. Identifying Traffic Differentiation on Cellular Data Networks. In *ACM SIGCOMM*, pages 119–120, 2014.

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

[48] A. Odlyzko. Paris Metro Pricing for the Internet. In *ACM EC*, pages 140–147, 1999.

[49] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. *IEEE/ACM Trans. Netw.*, 2(2):137–150, Apr. 1994.

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

[51] 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.

[52] J. Roberts. Internet Traffic, QoS, and Pricing. *Proceedings of the IEEE*, 92(9):1389–1399, Sept 2004.

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

[54] A. Singla, B. Chandrasekaran, P. B. Godfrey, and B. Maggs. The Internet at the Speed of Light. In *ACM HotNets*, pages 1:1–1:7, 2014.

[55] J. B. Speta. Handicapping the Race for the Last Mile: A Critique of Open Access Rules for Broadband Platforms. *Yale Journal on Regulation*, 17:39, 2000.

[56] J. B. Speta. Book Review: A Vision of Internet Openness by Government Fiat the Future of Ideas: The Fate of the commons in a connected world by Lawrence Lessig. *Northwestern University Law Review*, 96:1553–1607, 2002.

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

[58] B. van Schewick and M. N. Weiland. New Republican Bill Is Network Neutrality in Name Only. *Stanford Law Review Online*, 67:85, 2015.

[59] 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.

[60] T. Wu. Network Neutrality, Broadband Discrimination. *Journal of Telecommunications and High Technology Law*, 2:141, 2003.

[61] T. Wu. The Broadband Debate: A User's Guide. *Journal of Telecommunications and High Technology Law*, 3(69), 2004.

[62] T. Wu. Why Have a Telecommunications Law-Anti-Discrimination Norms in Communications. *Journal on Telecommunications and High Technology Law*, 5:15, 2006.

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

[64] C. S. Yoo. Vertical Integration and Media Regulation in the New Economy. *Yale Journal on Regulation*, 19:171, 2002.

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

[66] Y. Zhang, Z. M. Mao, and M. Zhang. Detecting Traffic Differentiation in Backbone ISPs with NetPolice. In *ACM IMC*, pages 103–115, 2009.

[67] Z. Zhang, O. Mara, and K. Argyraki. Network Neutrality Inference. In *ACM SIGCOMM*, pages 63–74, 2014.

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

Appendix

A. FIFO

FIFO is the simplest and dominant service policy currently deployed in the Internet. In the legal debate about network neutrality, FIFO scheduling is typically above all suspicion: It does not prefer (or discriminate against) particular users, protocols, or applications. On the other hand, FIFO does not enforce any notion of fairness, because it is commonly assumed to assign the forwarding resource proportional to arrival rates, i.e., it has low throughput interference. In fact, this is the essence of its perceived neutrality.

We investigate the intuitive, but to our knowledge, unproven conjecture that a *finite* FIFO buffer has low throughput interference, i.e., $TI^2 \approx 0$. The analytical discussion neglects traffic correlations, thus restricting it mainly to Poisson traffic.

For ease of presentation, let us consider two traffic flows accessing a FIFO buffer. Let $A_1^{in}(t)$, $A_2^{in}(t)$ be the cumulative arrivals up to time t from Flow 1 and 2, respectively; further, denote with $A_i^{out}(t)$ and $A_i^{loss}(t)$ the cumulative arrivals of Flow i up to time t that were accepted by the system, respectively not able to enter the buffer, such that $A_i^{in}(t) = A_i^{out}(t) + A_i^{loss}(t)$. The following (long-term) rates are defined

$$\lambda_i^{\bullet} = \lim_{t\to\infty} \frac{A_i^{\bullet}(t)}{t} \qquad \bullet \in \{in, out, loss\},\ i \in \{1, 2\}\,.$$

As a mathematical technicality, under a stochastic process interpretation of $A_i(t)$, the limits are taken according to mean-square convergence (see [24], Chapter 7). For the case of two CBR flows sharing an *infinite* FIFO buffer, Ghiassi-Farrokhfal et al. [22] show that under overload the two CBR flows share the server proportionally to their input rates, i.e.,

$$\frac{\lambda_1^{out}}{\lambda_2^{out}} = \frac{\lambda_1^{in}}{\lambda_2^{in}}\,,$$

and thus FIFO perfectly achieves $TI^2 = 0$. However, the analysis critically relies on the buffer to be infinite. In fact, for *finite* buffers it is not difficult to construct a pathological example to show that being "in sync" with the FIFO scheduler helps to dramatically increase a flow's share of the overall capacity, such that $TI^2 \to \frac{1}{2}$, the worst-case for two flows. Therefore, stochastic arrivals are considered. The following proposition establishes that for Poisson flows in a finite FIFO buffer the relative throughputs are equal in expectation and thus it can be expected that $TI^2 \approx 0$.

Proposition 3: If A_1, A_2 are independent Poisson arrival processes with parameters λ_1^{in} and λ_2^{in} then

$$\mathbb{E}\left(\frac{\lambda_1^{out}}{\lambda_1^{in}}\right) = \mathbb{E}\left(\frac{\lambda_2^{out}}{\lambda_2^{in}}\right)\,.$$

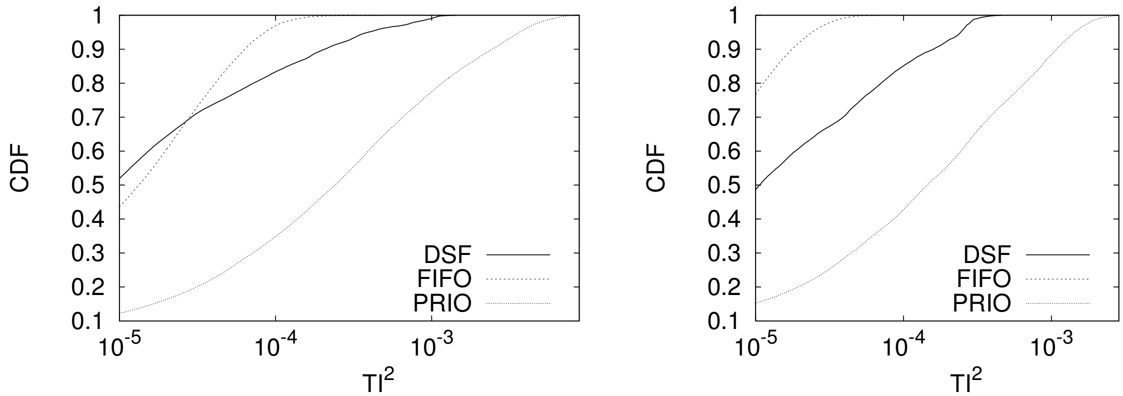
Proof: The time evolution of the system consists of an alternating sequence of periods when the buffer is not yet full and when it is full. Periods where the buffer is not yet full are termed *success periods* and denoted by $S_{1,}S_2,\ldots,$ as packets arriving are accepted by the system and will eventually contribute to the output for their corresponding flows. Periods where the buffer is full are called *loss periods*, denoted by $L_1, L_2, \ldots,$ as packets arriving are lost. Hence, time is partitioned into the sequence $S_1, L_1, S_2, L_2, \ldots$ Focusing on success periods, let $Y^i_{S_j}$ be the amount of packets arriving from flow i during success period S_j. A basic, though important, observation is that no matter when a certain success period starts, i.e., when the buffer becomes free, for both flows $Y^i_{S_j}$ is Poisson-distributed with parameter $\lambda_i^{in}|S_j|$, where $|S_j|$ denotes the (random) duration of S_j. This is due to the memoryless property of the exponential inter-arrival times for Poisson flows (in terms of renewal theory, at the start of the success period both flows have residual life that is exponentially distributed with parameter $\frac{1}{\lambda_i^{in}}$).

Using a slightly informal notation, the expected output rates for both flows ($i = 1, 2$) can be partially calculated as follows. $E_X(.)$ denotes the expectation operator with respect to random variable X.

$$
\begin{aligned}
\mathbb{E}\left(\lambda_i^{out}\right) =& \mathbb{E}_{S_j,L_j}\left(\mathbb{E}_{Y^i_{S_j}}\left(\lim_{k\to\infty}\frac{\sum_{j=1}^k Y^i_{s_j}}{\sum_{j=1}^k s_j + l_j}\middle| \begin{array}{c}|S_j| = s_j\\ |L_j| = l_j\end{array}\right)\right)\\
=& \mathbb{E}_{S_j,L_j}\left(\lim_{k\to\infty}\frac{\sum_{j=1}^k \mathbb{E}_{Y^i_{S_j}}\left(Y^i_{S_j}\middle| \begin{array}{c}|S_j| = s_j\\ |L_j| = l_j\end{array}\right)}{\sum_{j=1}^k s_j + l_j}\right)\\
=& \lambda_i^{in}\mathbb{E}_{S_j,L_j}\left(\lim_{k\to\infty}\frac{\sum_{j=1}^k s_j}{\sum_{j=1}^k s_j + l_j}\right).
\end{aligned}
$$

In the first line, the law of total expectation is applied. In the second line, the exchange of limit and expectation is possible because the convergence of the output rates is in the mean-square sense. Also, the linearity of the expectation operator is used. In the third line, the fact that $Y^i_{S_j}$ is Poisson-distributed is applied. Thus, the ratio of the output rates yields the proposition. ■

The restriction to two flows is without loss of generality due to the superposition property of Poisson flows. However, the memoryless property of the Poisson distribution is critical for the proof of Proposition 3. Hence, the proposition does not necessarily hold for general renewal processes (with arbitrary i.i.d. inter-arrival distributions), because the distribution of residual life is different from a fresh renewal epoch and depends on the variance of that distribution. Yet, from the Palm-Khintchine Theorem ([29], p.250) it is known that an aggregate of renewal processes (each with a small intensity) converges to a Poisson process and, therefore, these deviations should not be too significant. However, traffic correlations, especially long-range dependencies (LRD), are not covered by the theorem. Therefore, the FIFO analysis is complemented by simulation experiments with LRD traffic reported in Section VI-A.



(a) TI^2 over 2s interval

(b) TI^2 over 4s interval

Fig. 11: TI^2 Distribution

An interesting observation is that the the proof of Proposition 3 only depends on the unbiased acceptance policy of a finite buffer, rather than the FIFO service policy itself. The actual order in which packets are served from the buffer does not change the fact that a finite buffer with unbiased packet acceptance achieves a low throughput interference in the long term ($t \to \infty$). This even holds true for strict priority service, but it is expected that FIFO achieves a lower TI^2 on shorter time scales. The dependency of TI^2 on the scope of the time interval is illustrated by showing the results from Section VI-A, Figure 6a (1s window) for increasing sliding window sizes (2s, 4s) in Figure 11. TI^2 values become smaller with an increasing window size and trend towards 0 for all three schedulers.

B. Shared Queue

To illustrate the trade-offs associated with a small shared queue, a FIFO Tail Drop queue with buffer sizes between 1ms and 750ms is observed using packet-level simulation (cf. Section VI). For each buffer size, Figure 12a shows the maximum delay experienced by a delay-sensitive application and the mean goodput of file downloads as a scatter plot. The highest per-download goodput in this experiment is about 3 Mbit/s for 20 concurrent downloads and a propagation delay of 30ms; the corresponding delay lies above 200ms. If the buffer is configured to achieve a 100ms delay target, the downloads see a 12% reduction in goodput. If the delay target is 50ms, the goodput decreases by more than 32%. Figure 12a also shows that the trade-off for the 99 delay percentile is similar to the maximum delay case.

A second experiment studies CoDel's trade-off under a maximum delay (and 99 delay percentile) metric. Keeping CoDel's default parameters and adjusting its *TargetDelay* does not result in significant delay reductions, so different *Interval* parameters are evaluated between 1μs and 500ms [38], [47]. Figure 12b shows the corresponding results for the delay over the mean per-download goodput. Unlike the Tail Drop results, CoDel's trade-off for the 99 percentile is better than for the maximum delay. Nevertheless, achieving a 99 delay percentile of 50ms costs about 20%

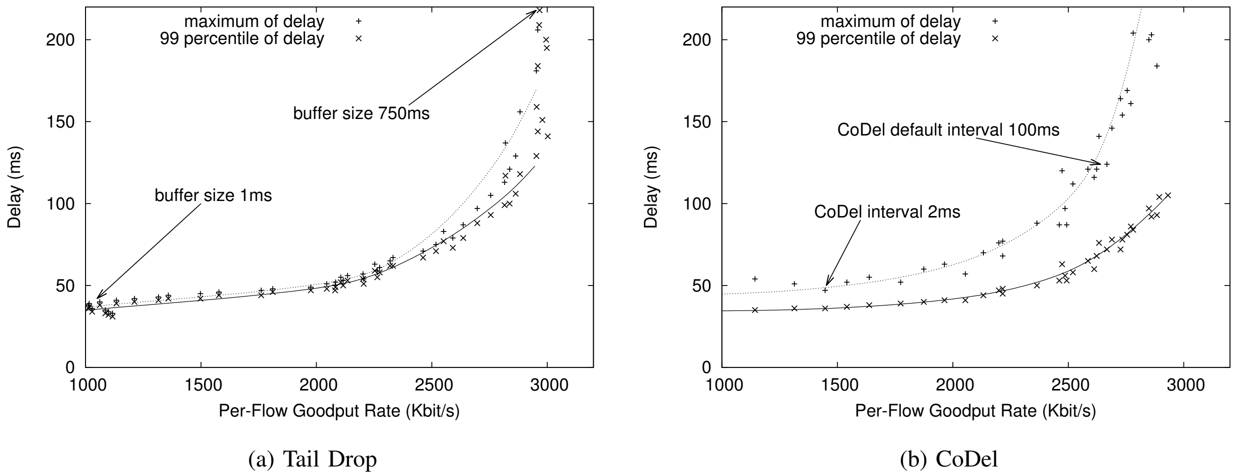


Fig. 12: Trade-off between Goodput and Low Delay

in goodput. Achieving a 50ms maximum delay is almost impossible when only adjusting the *Interval* parameter. Therefore, an exhaustive search in CoDel's parameter space was performed. This leads to an improved trade-off, in particular for low delay targets. However, a 50ms maximum delay target still decreases the file download goodput by about 36%. These findings confirm that aggregate delay control for low-delay targets entails a significant cost on goodput.

C. Global DD Markov Chain

As has been discussed in Section V-A, the N-dimensional state vector of a global Markov chain has entry values corresponding to: assigned (a), uncertain (u), and empty (e) slots. As the DD slot queue is a FIFO queue, the empty slots are always trailing. Assigned and uncertain slots are in the leading entries and can take any permutation over these. The number of uncertain slots is bounded by $N - N_1$. Under these conditions, the state vector can be represented by the regular language over the alphabet $\{a, u, e\}$ defined by

$$\{a^{i_k} u^{j_k} e^n \mid \forall k \geq 0 : i_k, j_k \geq 0, n \geq 0,$$

$$n + \sum_k i_k + j_k = N, \sum_k j_k \leq N - N_1\}.$$

In particular, the size of the state space of the global Markov chain can then be calculated as

$$\sum_{i=0}^{N} \sum_{j=0}^{\min\{N-N_1, N-i\}} \binom{N-i}{j}.$$

While no closed-form is known for such a sum of binomial coefficients, it is clear that a state space explosion is experienced for typical parameter settings. For example, $N = 1000$, $N_1 = 100$ yields $\sim 2.143 \times 10^{302}$ states.