# FIFO Service with Differentiated Queueing

Martin Karsten
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1, Canada
mkarsten@uwaterloo.ca

## ABSTRACT
This paper presents a novel approach to minimally invasive service differentiation in packet-switched networks. Instead of actively managing service allocation, a simple differentiated queueing algorithm provides traffic classes with essentially the same best-effort service that would result from plain FIFO service using a single queue for all traffic. However, each class is served from a separate virtual queue, which is configured with an individual deterministic delay bound that is enforced in the presence of dynamically varying packet arrival rates. The main advantage of such a scheme for service differentiation is administrative simplicity, because it only needs minimal configuration by a network operator and does not necessarily require control plane functionality. Further, it does not inherently prefer some traffic classes over others and thus satisfies even the most radical definitions of network neutrality. In the paper, the basic approach is motivated with the help of various use case scenarios. A fairly simple and efficient algorithm is presented to implement the differentiated queueing scheme. Finally, a number of simulation experiments and results are shown that confirm the intuitive functionality of the algorithm.

## Categories and Subject Descriptors
C.2.1 [**Network Architecture and Design**]: Packet-switching networks; C.2.6 [**Internetworking**]: Routers

## 1. INTRODUCTION
Consider a home network with two concurrently active applications: a large file transfer and a VoIP session. For a file transfer to or from a well-connected Internet server, the residential link might very well be the throughput bottleneck. In this scenario, the router at either end of the residential link (depending on the direction of the file transfer) must have buffering capacity in the amount of the file transfer's round-trip time (RTT) to ensure smooth TCP service and high link utilization. On the other hand, the quality of the VoIP session would be seriously degraded by the resulting queueing delay. Given the rate of adoption of conventional QoS mechanisms and their administrative overhead and configuration complexity, it is highly unlikely that this problem can be remedied easily. In fact, there are recent reports of a so-called *bufferbloat* phenomenon [19] that aggravates the fundamental buffering mismatch between sliding window-based reliable transfer and latency-sensitive interactive applications. In this paper, a very simple mechanism is presented that permits users and applications to choose from a small variety of delay classes in a way that does not result in inherently preferential service to any class. Thus, in the residential scenario, both file transfer and VoIP applications would pick different delay classes and experience appropriate buffering behaviour at routers, but without further service differentiation. Aside from this residential scenario, there are other deployment configurations where such a queueing regime would useful.

In general, different types of applications with different traffic profiles require different levels of buffering in Internet routers. For example, bulk data transfer using TCP needs a certain level of buffering at routers to effectively utilize communication links and generally does not exhibit quality degradations with larger buffers (up to a certain point). On the other hand, conversational applications (e.g., voice or gaming) do not benefit from large buffers, but instead their quality suffers when queues fill up and packets are delayed. While recent results give some indication that this quandary might be less pronounced in lightly loaded core networks with a high level of multiplexing (see Section 2.2.1 for details), it is still clearly an open challenge for routers at the edges of networks, such as home routers, ISP routers, and possibly public peering points.

The goal of *Virtually Isolated FIFO Queueing* (VIFQ) is to emulate FIFO throughput, but to also support differentiated strict delay classes at routers. VIFQ adopts a policy-free point of view where all arriving traffic is treated as equally important and valuable, thus leaving rate allocation decisions to other network components. Instead of actively managing service rates, VIFQ constructs multiple virtual FIFO queues that are configured with a maximum queueing delay each. Aside from packet drops to enforce the strict queueing delay targets, different classes are served in proportion to their respective arrival rates, similar to best-effort shared FIFO service. Therefore, each traffic source has an incentive to pick the service class with the most suitable buffer/delay configuration for its utility and burst characteristics.

This paper is the first to a) formulate this idea, b) present a feasible algorithm for a queueing scheme, and c) evaluate its characteristics. The paper is organized as follows. The general general idea and possible deployment scenarios are introduced in next section. Related work is discussed in Section 3. In Section 4, the queueing algorithm is presented and studied. The operation of the algorithm is demonstrated by means of simulations in Section 5. The paper is wrapped up with a summary discussion and conclusions in Section 6.

## 2. MOTIVATION
An intuitive conjecture about FIFO scheduling is its *rate neutrality*, i.e., the notion that the service rates of multiple traffic classes traversing a shared FIFO buffer are generally proportional to their respective arrival rates. While not proved in the most general case, there is some empirical [34] and analytical [11, 20] evidence, aside from intuition, to use it as a working hypothesis. Based on this

hypothesis, edge-based traffic control (such as TCP's congestion control algorithm) can manage network resources without any dynamic state or explicit signalling with routers. However, traffic and service management in this case only extends to rate allocation and does not include differentiated delay control. The goal of VIFQ is to add delay control and differentiation without changing the inherent characteristics of FIFO scheduling. It is a queue management module with the following properties:

- Support for multiple strict delay classes.
- No inherently preferential treatment of any traffic class.
- Best-effort service approximating shared FIFO service.
- No dynamic state or signalling.
- Efficient implementation at high line rates.

However, VIFQ does not address the inherent RTT-dependency of the TCP congestion control algorithm. In particular, TCP's sending rate is inversely proportional to the RTT, so delay differentiation indirectly affects end-to-end throughput for competing TCP sessions. Other proposals have attempted to address these effects with arguable success. This aspect is further discussed in Section 3 and experimentally studied in Section 5. It is important to note though that end systems are free to choose any delay class for their traffic, since there is no inherently preferential treatment. Thus, any perceived unfairness can be remedied by end systems choosing appropriate delay classes.

Given its objectives, VIFQ follows a laissez-faire approach where all traffic is transmitted as equally and optimistically as possible. Routers are statically configured with multiple delay classes and drop packets from a traffic class as needed to meet the respective delay target of that class. VIFQ does not create service classes with inherently higher throughput and, therefore, does not need any direct traffic management. It utilizes all available resources and does not require signalling with routers. It does not interfere with rate control by an external control regime and a control system can employ any pro-active traffic control, such as prioritization, marking, scheduling, policing, or admission control, in addition to using VIFQ.

The design of VIFQ is based on the observation that traffic profiles from different applications differ in their burstiness and delay targets. For example, interactive applications often prefer a smaller worst-case delay in exchange for a slightly higher drop rate. For this type of traffic it is beneficial to limit the maximum buffer at routers. In other words, during transient overload, those applications would rather have a few more packets dropped, than experiencing an unacceptable delay for all packets. One immediate complication is given by the dynamics of arrival and service rates. From an application perspective the size of the forwarding buffer in bytes is largely irrelevant. What matters is the maximum queueing delay in time units. In a multi-class scenario with dynamic rates for each class, this prohibits the use of static buffer limits. Instead, each per-class buffer has to be adapted to the corresponding service rate to ensure a constant maximum per-class queueing delay.

## 2.1 Conceptual Design

The simplest way of thinking about VIFQ is to start with regular FIFO scheduling, but add a simple delay test at service time. If a packet violates its deadline, it is discarded. However, in its stead, a packet from the same class is forwarded, if possible, so that the overall throughput rates of multiple classes stay as close to FIFO service as possible.

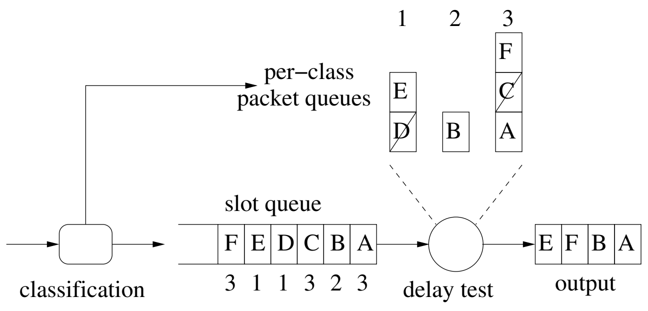

**Figure 1: Conceptual Design – Packets C and D too late**

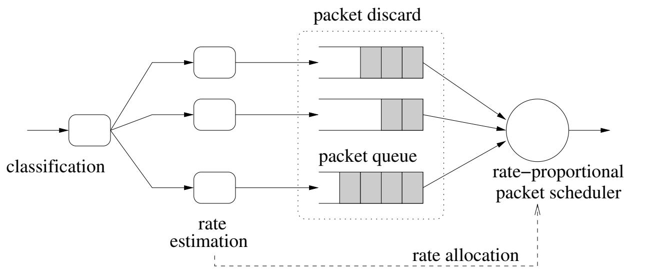

**Figure 2: Alternative Conceptual Design [27]**

The conceptual operation of VIFQ is shown in Figure 1. Traffic classes are serviced according to their arrival pattern. If, at departure time, a packet violates its target delay, it is dropped and the next packet from the same class is served. This shifts all subsequent packets for this class and the corresponding service is not lost while the class has packets in the queue. The main service queue stores class descriptors representing send slots, while per-class queues hold the actual packets. For example, in Figure 1, Packet C is found violating the target delay of Class 3 and is dropped. Instead, Packet F is moved ahead and forwarded immediately. Similarly, Packet D violates the target delay of Class 1 and is replaced by Packet E.

The pseudo code for the arrival and service routines is shown in Algorithm 1 and 2. For clarity reasons, the pseudo code shown here does not account for varying packet sizes. The second column in Table 1 summarizes the variables of the conceptual algorithm, while the third column applies to the actual algorithm presented in Section 4.

The arrival routine stores a class descriptor in the global slot queue $q_S$ and places the packet with deadline $d_p$ in a per-class packet queue $q_c$. The admission test in Line 3 does not consider the length of the arriving packet to avoid a bias against larger packet sizes. The service routine processes class entries in the slot queue and for each class, finds the next packet that can be sent without violating the class's target delay. Both loops in the service routine might execute several times. If packets are found violating the deadline, the inner loop executes several times. If a per-class packet queue runs empty, but the main slot queue has still entries for this class, the outer loop will execute more than once.

**Table 1: Parameters and Variables**

| Name | Conceptual Version | Actual Algorithm |
|---|---|---|
| $B$ | queue limit | queue limit |
| $b$ | backlog | virtual backlog |
| $t$ | arrival time | arrival time |
| $p$ | current packet | current packet |
| $l_p$ | packet size | packet size |
| $d_p$ | packet deadline | N/A |
| $c$ | service class | service class |
| $D_c$ | class delay target | class delay target |
| $q_c$ | class packet queue | class slot queue |
| $t_c$ | N/A | timestamp at head of $q_c$ |
| $l_c$ | N/A | packet size at head of $q_c$ |
| $x_c$ | N/A | class transmit credit |
| $q_S$ | slot queue | slot queue |
| $t_S$ | N/A | timestamp at head of $q_S$ |
| $l_S$ | N/A | packet size at head of $q_S$ |
| $c_S$ | N/A | class at head of $q_S$ |
| $q_P$ | N/A | packet queue |

Unfortunately, there is a caveat with this simple design. The runtime of the service routine cannot be limited effectively, since it is impossible to predict how many consecutive packets will miss their deadline and/or how many slots are left unused. In theory, there is clearly some upper bound based on the minimum packet size and the total amount of buffer. However, given the tight timing with which packets must be released to a high-speed output link, it is not possible to ensure that the output link is always fully utilized when the runtime of the service routine cannot be tightly controlled.

A different way of conceptualizing the queueing algorithm is given by the design of ICDS [27] shown in Figure 2, where measured arrival rates are used to directly configure a rate-proportional scheduler. However, ICDS does not provide a feasible algorithm and is included here only for illustration.

Intuitively, this queueing and delay control scheme provides service very close to a regular shared FIFO queue, but at the same time, strictly enforces the delay target of each service class. A packet that violates its deadline is replaced by another packet of the same class, which results in an overall service shift for all current and future packets in that class during the same busy period. This minimizes the impact of delay drops and the relative throughput of each class does not deviate much from FIFO queueing. However, there is one exception. A class that receives traffic at a very low rate might be starved from service by not having another packet available in the per-class queue $q_c$ at the time when a previous packet is discarded in the service routine. Such traffic must be placed in a *default class* that is configured with a target delay equal to the maximum system queueing delay. The actual VIFQ algorithm is studied in more detail in Section 4.2.

## 2.2 Use Case Scenarios

Different applications and usage scenarios vary greatly in their requirements for transmission rate. Many applications can operate successfully within a spectrum of different throughput rates. In economic terms, such applications have a concave rate utility function for throughput. Therefore, a freely configurable transmission rate (as opposed to a few fixed rate classes) is highly desirable for a multi-service communication network.

**Algorithm 1** Conceptual Arrival Routine

```
 1: $p \leftarrow$ received packet;
 2: $c \leftarrow$ service class of $p$;
 3: **if** $b < B$ **then**
 4:     tail($q_c$) $\leftarrow p$;
 5:     tail($q_S$) $\leftarrow c$;
 6:     $b \leftarrow b + l_p$;
 7:     $d_p \leftarrow t + D_c$;
 8: **else**
 9:     drop($p$);
10: **end if**
```

**Algorithm 2** Conceptual Service Routine

```
 1: **while** $q_S$ not empty **do**
 2:     $c \leftarrow$ head($q_S$);
 3:     **while** $q_c$ not empty **do**
 4:         $p \leftarrow$ head($q_c$);
 5:         $b \leftarrow b - l_p$;
 6:         **if** $t \leq d_p$ **then**
 7:             $t \leftarrow t + l_p$;
 8:             transmit($p$);
 9:         **else**
10:             drop($p$);
11:         **end if**
12:     **end while**
13: **end while**
```

In contrast, it seems less important to support a wide variety of delay settings. In economic terms, delay utility curves are usually S-shaped, since delay requirements are based on human perception or other external factors. For similar reasons, there is not much variety in delay requirements of different applications and thus, a fixed number of global delay classes should be sufficient. Consequently, VIFQ can be envisioned as being deployed with a small fixed number of static delay classes, which can be selected by setting an appropriate code point in the packet header. The set of delay classes and code points can be standardized.

One caveat is that queueing delay is additive and thus the traversal of multiple bottlenecks could add up the actual delay to a multiple of the target delay. However, since VIFQ forwarding works with dynamic rates, this seems less of a problem compared to the fixed-rate scenario in reservation schemes such as the one analyzed by Parekh and Gallager [35] and subsequently others. Instead, it is assumed that most often a traffic flow experiences a single bottleneck along its path and thus, a significant queueing delay only applies once. Further, in most usage scenarios VIFQ would only be deployed at certain routers along an end-to-end path, which also limits the impact of additive delay.

With these considerations in mind, various use case scenarios are presented and discussed here to motivate the VIFQ proposal, as well as to illustrate avenues for future research.

### *2.2.1 Small Router Buffers*

Recently, it has been speculated that for TCP-style traffic flows the necessary amount of buffering in routers may be much less than previously assumed and may become irrelevant in terms of delay [29]. While this may be true for backbone routers [3], it is not clear whether this assumption always holds [30, 32, 37] and in particular, whether it holds for all parts of the network [14]. Also, while link utilization is reasonably high with a sufficient number of desynchronized TCP flows [5], there is no clear and direct explanation how TCP flows desynchronize under which circumstances. Further, an argument against small router buffers is the resulting increase in variability of TCP throughput per flow [13]. VIFQ can help addressing those concerns by supporting both small and regular buffer sizes at the same time and letting source nodes (or gateways) decide any potential trade-offs and experiment with different buffer sizes. One argument in favour of small router buffers is the resulting lower delay for all traffic, including delay-critical traffic. VIFQ offers an alternative and more flexible solution for that challenge by offering different service classes. However, another benefit of small router buffers applies to router design, especially in the optical domain. VIFQ does not address this concern directly, but might expedite research and in-situ experiments with small buffers.

### *2.2.2 Isolated Deployment*
When deployed in isolation, VIFQ can be a very useful building block, because it opens up new possibilities for end systems to influence their packets' treatment at overloaded links, but without interfering with existing network operations and management. VIFQ can be used in isolation at specific routers along the data path that are prone to overload, but where sophisticated traffic management is not feasible.

A good example are Internet peering points between network domains. If these exchanges are highly loaded, it might not be feasible to add extra control functionality. Further, inter-domain QoS inevitably requires even more complicated accounting and/or billing structures than single-domain offerings. Even in the absence of a globally coherent QoS system, VIFQ can facilitate choice and delay differentiation for different traffic flows from various applications.

Another example for using VIFQ is in edge routers near or on customer premises, as already briefly discussed at the beginning of Section 1. Because of the relatively small number of traffic flows in such edge routers, the rationale for small router buffers (see previous section) does not apply. Instead, routers need to be configured following the traditional recommendation that buffer sizes should be around $C \cdot RTT$, where C is the link capacity and RTT is the maximum round-trip time excluding the buffer [9, 43]. Since the round-trip time may be large, the buffer needs to be equally large. Without any service differentiation, traffic from interactive applications traverses the same queue and experiences unnecessary and potential harmful queueing delay, if the edge link is the bottleneck of a concurrently running TCP session. Although the computational complexity of differentiated packet scheduling schemes is limited with a small number of active flows, pro-active traffic management schemes are not a suitable choice in this scenario, because of their inherent throughput differentiation and/or administrative overhead. This is further discussed in Section 3.

VIFQ solves this problem by discarding some packets from a low-delay class when the queue builds up, but in exchange guarantees the queueing delay for all other packets in that class. Deploying VIFS with a small number of delay classes at such edge routers would solve the short-term problem of TCP-sized router buffers interfering with interactive applications. However, it would not interfere with the simple traffic management regimes that are currently found at the edge of the Internet. More importantly, it would still allow for future innovations and alternative traffic management schemes like, for example, those discussed in Section 2.2.4.

### *2.2.3 Active Queue Management*
*Active Queue Management* (AQM) schemes operate on a router queue by marking or dropping packets, before the queue is completely full. This is done to tighten and smooth the control loop between rate-controlled edge systems and network routers. In particular, AQM schemes are credited with reducing latency, improving fairness, and avoiding synchronization between flows. See [39] for an earlier survey of the field. VIFQ can be combined with any non-preemptive AQM scheme, i.e., any scheme where a packet is either discarded at arrival or not at all.

### *2.2.4 Edge-based Load Control*
In recent years, a class of edge-based rate and admission control schemes have been proposed [8, 16, 28, 29] that can provide differentiated services without dynamic state in core routers. These schemes operate based on load feedback that is conveyed to edge systems via simple packet marking, similar to TCP/ECN [38]. Edge-based traffic control can only manage network resources effectively, if core routers do not interfere with the rate allocation decisions at the edge. Consequently, these schemes work well with FIFO scheduling, which however does not support delay differentiation.

Edge-based load control systems operate on the premise that routers signal their current and/or incipient load situation to edge or end systems, which in turn make rate adaptation or admission control decisions to manage traffic and avoid long-term overload situations. An example for a system proposal along with further references is given in [16]. This work is conceptually rooted in TCP's congestion control algorithm, which uses packet loss as a load signal, as well as more recent packet marking schemes, such as *Explicit Congestion Notification* (ECN) [38]. In terms of theoretic analysis, it is based mainly on work done by Kelly [29] and others. More involved proposals exist, such as XCP [28], which provide better control and efficiency at the cost of higher complexity. However, all proposals implicitly assume a single scheduling class and limit service differentiation to controlling the transmission rate. Further, in the dynamic case it seems impossible to reliably avoid transient overload situations under realistic assumptions [25], especially when high utilization is also sought. In fact, the IETF PCN architecture [16] requires a separate isolated traffic class for "PCN traffic" to meet low-delay goals.

Because VIFQ does not interfere with basic FIFO service rates, it is the ideal companion for such a control regime. Edge or end systems can follow through with their rate allocation schemes and sources can mark packets for a specific delay target, which is then enforced by VIFQ. This creates a simple yet effective multi-class QoS system without the need to label some traffic as special or more important.

## 3. RELATED WORK
Service differentiation in IP networks is a long-standing open issue. There are numerous technical proposals focusing on various aspects of the overall problem. Proposed solutions comprise differentiated scheduling at the flow or traffic class level, admission control, and eventually accounting. As such, they are always an amalgam of data plane (scheduling), control plane (signalling), and management (accounting) operations. Complexity may arise from each of these components and should be avoided as much as possible.

Traditional service differentiation schemes come in the form of rate control mechanisms and are based on either of two paradigms:

priority forwarding or resource allocation. In priority-based systems, such as Expedited Forwarding [12], high priority forwarding is used to keep delays small. Because of the resulting forwarding preference, traffic control and management is needed to keep the amount of high-priority traffic limited. In allocation-based systems, such as the gamut of proposals for Integrated Services [7], a maximum queueing delay can only be guaranteed, if the arriving traffic conforms to a specific traffic envelope [35]. In either case, the amount of queueing delay ultimately depends on the total amount of arriving traffic, which needs to be managed. Correspondingly, any scheduling method other than FIFO needs configuration, control, and accounting, all of which result in increased complexity. VIFQ is a much simpler approach that does not require external traffic management, control, or accounting. There might be interesting deployment scenarios including VIFQ and flavours of the above mechanisms, but VIFQ has no inherent complexity.

Traffic aggregation is an approach to limited the overhead of service differentiation using traditional schemes. Traffic is bundled into traffic aggregates and sufficient throughput capacity is allocated per aggregate in the core of the network to forward a predefined maximum traffic rate without exceeding queueing delay limits. There is considerable work that studies the relationship between rate allocation and delay guarantees for traffic aggregates [6, 10, 41]. However, in the presence of dynamic flow arrivals and traffic rates, a traditional multi-class traffic control system still has to resort to one of two basic control schemes for managing traffic while guaranteeing queueing delay:

- With fixed rate allocations, traffic has to be fitted to the rate allocation. This requires strict policing and might be inefficient, if other traffic classes do no fully utilize their rate allocation.
- Dynamic rate allocation requires signalling with core routers. While this does not directly violate the requirement for as little state information as possible, the resulting signalling interactions pose an essentially similar overhead to core routers.

Strict priority scheduling is a very simple form of service differentiation. In fact, it is one example for a relative or proportional class-based scheduler. For example, the *Best Effort Differentiated Services* (BEDS) [17] maintains fixed delay and drop ratios between two services classes. One of the service classes is intended for delay-sensitive traffic, but might incur a higher packet loss, while the other one provides lower packet loss. A similar proposal, although different in certain details has been presented as *Equivalent Differentiated Services* (EDS) [18]. In both cases, it is unclear how to configure the respective service ratios for any meaningful absolute service guarantee. A thorough theoretical treatment of relative average delay differentiation [15] works by adjusting the scheduler to traffic conditions under the assumption that packets are never dropped. This is fundamentally different from VIFQ, which guarantees absolute worst-case delays by potentially dropping packets. Generally, attempts at relative service differentiation at routers impose a throughput policy that might interfere with edge-based rate allocation. Another approach to relative service differentiation is given by so-called "lower-than-best-effort" approaches in the transport layer [44], but these scheme also do not explicitly address delay differentiation.

The *Alternative Best-Effort* (ABE) [21, 22] and *RD Service* [36] proposals each present a stand-alone two-class service model, where one class provides a deterministic bound on queueing delay, but does not adversely affect the throughput of the other class. Because traffic rates from TCP-type sources increase with reduced RTT, both proposals contain "fairness" provisions, such that a flow cannot attain higher throughput in the delay-oriented class compared to a similar flow that transmits using the throughput-oriented class.

In the ABE implementation, the efficiency of the admission test for delay-oriented packets is critically linked to managing only two services classes. In addition, this admission test does not completely eliminate the need for packets drops in the service routine, which in turn results in the same unbounded worst-case overhead as the conceptual design shown in Section 2.1. Further, the scheduling algorithm utilizes a parameter $g$ to choose between both classes, if current packet deadlines do not mandate a specific order of service, to mitigate the effect of smaller delays on feedback-controlled sources. However, this approach is based on the assumption that the RTT for all flows is identical and known. The computation of $g$ is not trivial and would only be complicated by supporting an arbitrary number of classes.

The RD Service proposal also offers two service classes only and requires an a-priori policy decision about the relative throughput rate targets. Its fundamental design trade-offs are very similar to ABE. Again, packets might need to be dropped during the service routine and enforcing the throughput policy requires a complex and error-prone estimation - in this case of the number of ongoing flows.

Both ABE and RD Service do not offer a generally feasible low-complexity scheduling algorithm. Also, both proposals couple delay differentiation with rate control. As such, they have a fundamentally different set of goals than VIFQ. In contrast, VIFQ separates both dimensions and therefore increases the flexibility to place arbitrary rate control functionality anywhere in the network - which of course can resemble ABE's or RD's throughput policies, if desirable.

Fundamentally, the deployment of ABE and/or RD Service would entrench a specific policy in core routers and interfere with edge-based rate allocation. This would ossify the Internet architecture and make it more difficult to experiment with alternatives to TCP's control algorithms. For example, Mathis [33] has recently questioned whether the narrow focus on "TCP-friendly" congestion control is still timely and adequate. RTT-"unfairness" is inherent to TCP's congestion control algorithm and should be addressed in that context, instead of hard-coded scheduling policies in routers.

One of the key aspects of VIFQ is that it permits the decoupling of rate and delay control. This must not be confused with the rate and delay decoupling offered by service curve schedulers, such as SCED [40] or L-SCED [4]. With a service curve scheduler, it is possible to configure rate and delay targets independently of each other – in contrast to, e.g., rate-based schedulers. However, the scheduling functionality is still tightly coupled in a single module and delay control requires strict policing. Further, these schedulers are inherently complex and there is no known algorithmic design that can execute at low constant complexity feasible for high-speed links. In contrast, VIFQ provides differentiated delay control and can be implemented in a form of constant complexity with low absolute overhead.

**Algorithm 3** VIFQ Arrival Routine

```
 1: $p \leftarrow$ received packet;
 2: $c \leftarrow$ service class of $p$;
 3: if $b < B$ then
 4:     tail($q_c$) $\leftarrow \langle t+b, l_p \rangle$;
 5:     tail($q_S$) $\leftarrow \langle t+b, c \rangle$;
 6:     $b \leftarrow b + l_p$;
 7:     $x_c \leftarrow x_c + l_p$;
 8: end if
 9: while $t_S < t + b - B$ do
10:     if $t_S = t_{c_S}$ then
11:         $x_c \leftarrow x_c - l_{c_S}$;
12:         $b \leftarrow b - l_{c_S}$;
13:         pop_head($q_{c_S}$);
14:     end if
15:     pop_head($q_S$);
16: end while
17: if $x_c \geq l_p$ and $t_c \leq t + D_c$ then
18:     $q_P[t_c] \leftarrow p$;
19:     $x_c \leftarrow x_c - l_p$;
20:     while $l_p > 0$ do
21:         if $l_p < l_S$ then
22:             $l_S \leftarrow l_S - l_p$;
23:             $l_p \leftarrow 0$;
24:         else
25:             $l_p \leftarrow l_p - l_S$;
26:             pop_head($q_c$);
27:         end if
28:     end while
29: else
30:     drop($p$);
31: end if
```

Dropping packets from the front of the queue has been suggested before and found to improve the overall queueing delay and thus responsiveness of the network [31]. However, the system studied there is single-class and the drop from front is triggered directly by the arrival of a new packet.

## 4. VIFQ

The VIFQ design closely emulates the conceptual design presented in Section 2.1, but avoids unlimited packet discard operations in the service routine. Instead of managing service rates, VIFQ differentiates between delay classes by using an appropriate queueing and dropping mechanism. VIFQ is illustrated in Figure 3, showing the same arrival pattern as in Figure 1. The arrival and service routines are shown in Algorithm 3 and 4. Note that for brevity Algorithm 3 uses $c_S$ directly as an index to $t$, $l$, and $q$ in Lines 10-13 to denote that the respective value is taken from the class at the head of $q_S$.

### 4.1 Algorithm

The system emulates a FIFO scheduler, similar to the conceptual design, but performs all critical operations in the arrival routine to manage the execution overhead. The key idea is that VIFQ manages *virtual slots* that represent packets in the comparable FIFO queue. Arriving packets that would be accepted into the comparable FIFO queue create a virtual slot in VIFQ. Virtual slots are tracked in the system and used for (potentially later) packets, such that a packet might be using the virtual slot of a packet previously discarded for violating its delay target. This emulates the service shift shown in the service routine of the conceptual scheduler in Algorithm 2. The arrival routine is comprised of the following components: *Admission control* creates virtual slots, while *delay test* uses a virtual slot for a packet, if that packet can then be sent without violating the delay target. The *cleanup* component deletes old and unused virtual slots from the system. Admission control and delay test are discussed first, followed by cleanup.

**Algorithm 4** VIFQ Service Routine

```
1: if $q_P$ not empty then
2:     $p \leftarrow$ head($q_P$);
3:     $b \leftarrow b - l_p$;
4:     $t \leftarrow t + l_p$;
5:     transmit($p$);
6: end if
```

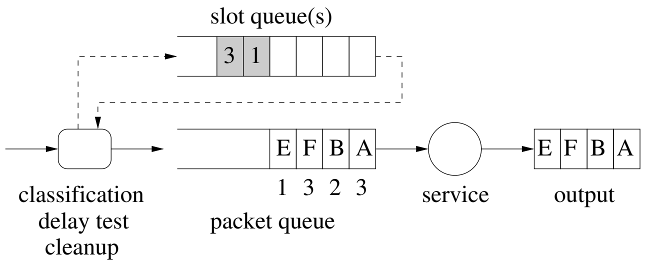

**Figure 3: VIFQ Algorithm – Packets C and D too late**

*Admission Control*

VIFQ maintains a queue of virtual slots that closely tracks the corresponding queue in a regular FIFO system. It first executes a global packet admission test (Line 3) - independent of the traffic class - similar to the conventional buffer test for a shared FIFO queue. First, this ensures a basic packet admission pattern similar to FIFO. Second, it limits the startup delay for a service class that has been dormant and begins receiving new packets, similar to the comparable FIFO queue. For each packet that passes the admission test, a virtual slot is stored in a per-class queue $q_c$ and a corresponding class entry is stored in the main slot queue $q_S$ (Lines 4,5). The virtual slots are labelled with a timestamp that represents the service time in the corresponding FIFO system. This timestamp is based on the existing length of the virtual queue, regardless of whether all virtual slots are eventually used for actual packets. Finally, the virtual backlog and per-class transmit credit are increased to reflect the arrival in the virtual queue (Lines 6,7).

*Delay Test*

The conceptual version performs the delay test in the service routine, immediately before a packet is being sent out. While this is the optimal place for the delay test in terms of accuracy, it also leads to an unlimited runtime for the service routine, which is a problem for high-performance execution, as discussed in Section 2.1. To eliminate this performance bottleneck, VIFQ performs the delay test during the arrival routine. The queue of virtual slots is conservative in that it holds virtual slots for all packets that would be accepted into the corresponding FIFO queue. Therefore, the timestamp assigned to a virtual slot in Lines 4 and 5 is an upper bound for the time when the class has the right to send a packet. The delay test is based on this conservative timestamp and the decision about a packet's acceptance is made by considering the oldest unused virtual slot that is available in the per-class queue. If that slot allows to send the currently arriving packet within the target delay bound

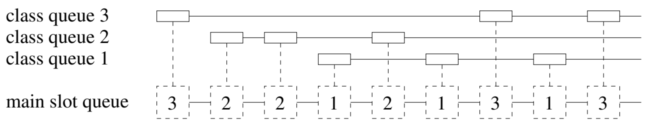

**Figure 4: Main and Class Slot Queues**

and the overall class transmit credit is large enough (Line 17), the packet is accepted for service (Line 18) at the given time. The code in Lines 19-28 reconciles varying packet sizes by borrowing transmit credit from later slots, if necessary. While this can slightly distort the delay bound, the error is clearly limited. For each class, less than one MTU worth of data can be sent ahead of time through this mechanism. Therefore, the overall error is limited to $N \cdot MTU$ where $N$ is the number of classes - which is expected to be rather small, as explained in Section 2.2.

*Cleanup*
One caveat of keeping virtual slots in the system is that "old" virtual slots might never be used, but still occupy virtual queue space. To avoid virtual buffer hogging by unused slots, VIFQ is designed to store only slots that cover a service time span equivalent to the overall buffer limit *B*, but this parameter might be subject to future investigation. For example, the choice of *B* here determines the highest arrival rate at which traffic might be starved (cf. Section 2.1). In this case, if the packet inter-arrival time of a class is larger than the maximum system queueing delay of $\frac{B}{\text{link speed}}$, then that class might not receive any service and corresponding traffic should be placed in the default class. Unused virtual slots are purged from the system (Lines 9-16) to approximate the removal of unused slots in the service routine of the conceptual version (cf. Algorithm 2). In particular, the test in Line 9 checks whether the current service time of the system ($t+b$) exceeds the service time of the oldest virtual slot ($t_S$) by the buffer size *B*. If that slot is unused (Line 10), it is deleted. In fact, cleaning up unused virtual slots is the only reason for the main slot queue's existence in VIFQ, because it facilitates access to slots in their global arrival order. The relationship between per-class queues and main slot queue is illustrated in Figure 4. The main slot queue is entirely managed by the cleanup procedure and consistency between both types of queues is maintained through timestamps (cf. Line 10).

## 4.2 Service Characteristics

VIFQ does not require any control or accounting and is thus best characterized by its game-theoretic properties. The following analysis is adapted from earlier work [27] and applies to open-loop traffic under overload conditions. Each traffic source is considered as one of *m players*. Each player *j* is defined by its maximum delay target $d_j^* \in D = \{d_1, \ldots, d_n\}$ where *D* is the set of delay targets provided by *n* service classes of the VIFQ *mechanism*. The delay target $d_j^*$ is called the *type* of player *j* and assumed to be private information (i.e., the mechanism and the other players do not know $d_j^*$).

A *strategy* for player *j*, $s_j$, is a mapping $s_j : D \mapsto D$, where, given its true delay target $d_j^*$, the player announces some delay target $s_j(d_j^*) = d_j$ by means of choosing the corresponding VIFQ service class. A strategy profile, $s = (s_1, \ldots, s_m)$, is a vector specifying that each player *j* is playing strategy $s_j$. A strategy profile is also written as $s = (s_j, s_{-j})$ where $s_{-j} = (s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_m)$.

The outcome of the game depends on the strategies chosen by the players. In particular, given a strategy profile *s*, the outcome is described by $o(s) = (d(s), \gamma(s))$ where $d(s) = (d_1(s), \ldots, d_m(s))$ specifies the delay $d_j(s)$ that the traffic of player *j* experiences, and $\gamma(s) = (\gamma_1(s), \ldots, \gamma_m(s))$ where $\gamma_j(s)$ is the drop rate experienced by player *j*.

The players have preferences over different possible outcomes, given their type $d_j^*$, which is captured using utility functions $u_j$. It is assumed that all the players are self-interested so that their utility depends only on the delay and drop rate of their own traffic. That is
$u_j(o(s), d_j^*) = u_j((d(s), \gamma(s)), d_j^*) \equiv u_j(d_j(s), \gamma_j(s), d_j^*)$.

Each player tries to choose a strategy that maximizes its utility, given that all other players are doing the same.

A strategy is said to be dominant, if it is a player's best strategy against any set of strategies that other players may choose. Formally, a strategy $s_j^*$ is a dominant strategy if $u_j(o(s_j^*, s_{-j}), d_j^*) \geq u_j(o(s_j, s_{-j}), d_j^*)$ holds for all $s_j \neq s_j^*$ and arbitrary $s_{-j}$. A *dominant strategy equilibrium* is a strategy profile $s^*$ where every player is using a dominant strategy. While many games do not have dominant strategy equilibria, VIFQ has one. In fact, the players' dominant strategies are to reveal their true target delay to the mechanism. Based on the following assumptions about the system and the players' utility functions, VIFQ is *strategy-proof*, which is an even stronger notion than *incentive-compatible*.

1. If $d_j > d_k$ then $\gamma_i(d_j, s_{-i}) \leq \gamma_i(d_k, s_{-i})$.
2. If $d_j(s) > d_j^*$ then $u_j(o(s), d_j^*) = 0$.
3. For two strategy profiles $s$ and $s'$, if $d_j(s), d_j(s') \leq d_j^*$ and $\gamma_j(s) = \gamma_j(s')$ then $u_j(o(s), d_j^*) = u_j(o(s'), d_j^*)$.
4. For two strategy profiles $s$ and $s'$, if $d_j(s), d_j(s') \leq d_j^*$ and $\gamma_j(s) \leq \gamma_j(s')$ then $u_j(o(s), d_j^*) \geq u_j(o(s'), d_j^*)$.

Assumption 1 states that a lower delay target, enforced by a smaller virtual buffer, results in a higher drop rate. Assumption 2 states that if the queueing delay is greater than the maximum target delay then the player's utility is 0. Assumption 3 states that for a fixed drop-rate, then as long as the delay is less than the maximum target delay, the player is ambivalent between the two outcomes. Assumption 4 states that as long as the delay is less than the target delay, a player prefers to have a lower drop rate.

THEOREM 1. *VIFQ is a strategy-proof mechanism.*

PROOF. (Sketch) If a player declares its true target value, that is follows a strategy $s_j^*(d_j^*) = d_j^*$ when all others play $s_{-j}$, then its utility will be $u_j(o(d_j^*, s_{-j}), d_j^*) \geq 0$. If, instead, the player had followed strategy $s_j(d_j^*) > d_j^*$ then its traffic would be in a class with delay greater than its true target, resulting in $u_j(o(s_j, s_{-j}), d_j^*) = 0$ (Assumption 2) and so the player is better off declaring its true target value. If the player had followed a strategy $s_j(d_j^*) < d_j^*$ then its traffic would be in a class with lower delay lower, but with a higher drop rate (Assumption 1). From Assumptions 3 and 4 follows that the player would be better off by declaring its true target value. Since *j* and $s_{-j}$ were arbitrarily chosen, any player is always best off revealing its true target delay. □

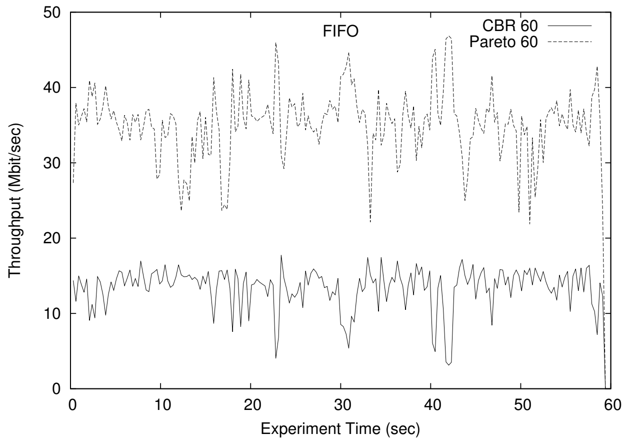

**Figure 5: FIFO Throughput - 30% CBR, 80% Pareto**

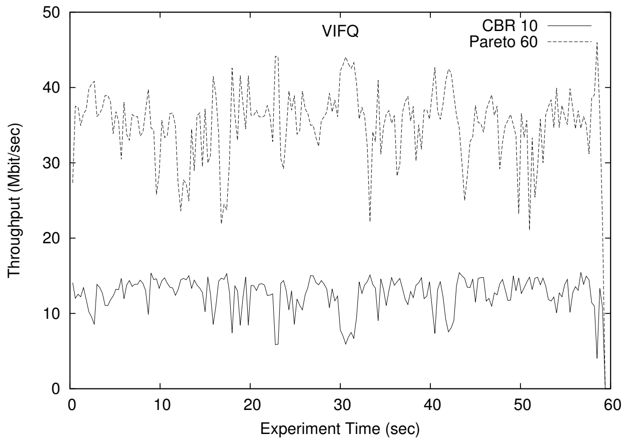

**Figure 6: VIFQ Throughput - 30% CBR, 80% Pareto**

## 4.3 Execution Overhead

The VIFQ algorithm does not fundamentally deviate from the complexity of a regular FIFO scheduler, except for two non-trivial aspects that are discussed below. The limited execution overhead of VIFQ is evident from the pseudo-code, which essentially maintains a few counters and linked lists. The key aspect is to perform all non-trivial operations in the arrival routine to better control the execution overhead.

A router or switch is usually designed to handle a worst-case traffic load of only minimum-sized packets. The two loops in the arrival routine (Lines 9-16 and 20-28) of VIFQ execute in proportion to the size of the currently arriving packet. Consequently, the execution overhead of the arrival routine is directly proportional to the size of the arriving packet. If an underlying hardware platform is capable of handling minimum-sized packets at arrival speed using FIFO scheduling, it will also be able to perform the VIFQ arrival routine on arbitrarily sized packets at arrival speed. This principle is termed *packet-amortized constant complexity* [26].

The only other potentially costly operation is maintenance of the packet queue $q_P$, which is sorted by the estimated service time of virtual slots and requires random-access insertion. It cannot be efficiently implemented as a linked list, but there are at least two feasible alternatives.

The first option is using a timer wheel [42], augmented by a hierarchical bitmask and using a *find-first-set* (ffs) operation to find the next element. The ffs operation finds the position of the least significant bit in a word. It can be implemented in software at logarithmic cost in the word length [1] or as a hardware priority encoder operating at low cycle cost. For example, Intel processors execute this operation in 1-3 clock cycles, depending on the architecture [23, 24]. As an example, a two-level hierarchy of 32-bit words can cover 1024 time slots. Assuming a total buffer space equivalent to 250 milliseconds, each slot covers roughly 250 microseconds. This accuracy is sufficient for packet deadlines.

On the other hand, as explained in Section 2.2, only a moderate number of delay classes are needed for deploying VIFQ. Therefore, a more traditional implementation of a class-based priority queue might also suffice, which would typically result in logarithmic overhead in the number of classes.

## 5. EVALUATION

The VIFQ algorithm emulates shared FIFO queueing as closely as possible, but with strict delay control and differentiation. The basic algorithm and its service characteristics are fairly straightforward. VIFQ has been implemented in the ns-2 simulation environment [2], following the pseudo-code given in Section 4.1. The main goal of the simulation experiments presented here is to confirm that VIFQ differentiates delay properly and to assess the throughput deviation between FIFO and VIFQ service in a controlled setting.

Experiments are carried out in a simulated dumbbell topology with a dedicated pair of sender and receiver nodes for each traffic flow. Except where noted, the following default configuration is used: The bottleneck link is configured with 50 Mbit/s. All access links are configured with twice the bottleneck speed. All link propagation delays are set to 10 milliseconds, i.e., the round-trip propagation delay is 60 milliseconds. All packet sizes are set to 500 bytes. Each experiment runs for 60 seconds of simulated time and is repeated 50 times with different seeds. The average results are reported here and unless the standard deviation is larger than 5% of the average, it is omitted in the figures.

## 5.1 CBR vs. Pareto Sources

In the first simple experiment, a constant bit rate (CBR) source transmits concurrently with a set of 32 Pareto sources with a shape parameter of 1.4, which produce self-similar background traffic. The offered load from the CBR source varies between 20% and 90% of the bottleneck capacity. The aggregated offered background load conversely varies from 90% to 20% for a combined expected average load of 110% of the bottleneck link. The experiment compares a FIFO queue with a buffer size of 60 milliseconds (msec) with a two-class VIFQ configuration where the CBR traffic uses the 10 msec class and the background traffic uses the 60 msec class. As a first illustration, Figures 5–8 show an example experiment run for 30% CBR traffic. Judging from a visual inspection of the results, VIFQ delivers the service that is intended. Figures 5 and 6 show that the rate allocation under VIFQ tracks FIFO service very closely. On the other hand, Figures 7 and 8 show that the maximum queueing delays are nicely differentiated, with the 60 msec class tracking the maximum delay under FIFO, while the 10 msec delay target for CBR traffic is also met when using VIFQ.

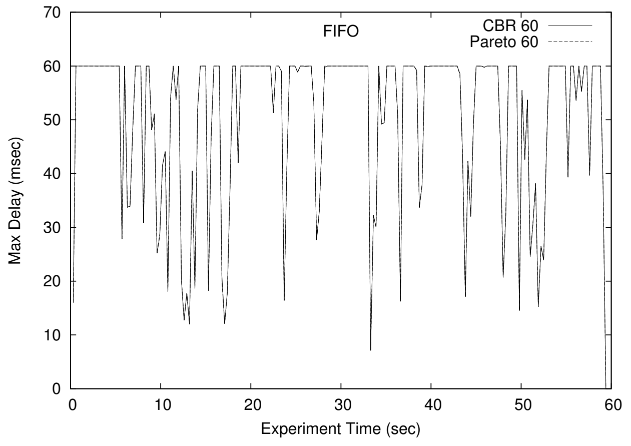

**Figure 7: FIFO Delay - 30% CBR, 80% Pareto**

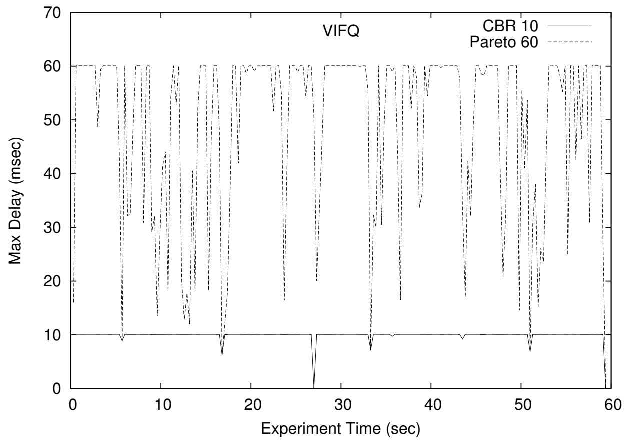

**Figure 8: VIFQ Delay - 30% CBR, 80% Pareto**

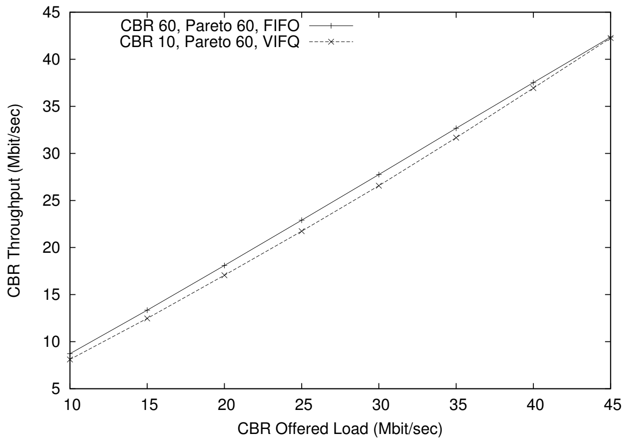

**Figure 9: CBR Throughput - FIFO vs. VIFQ**

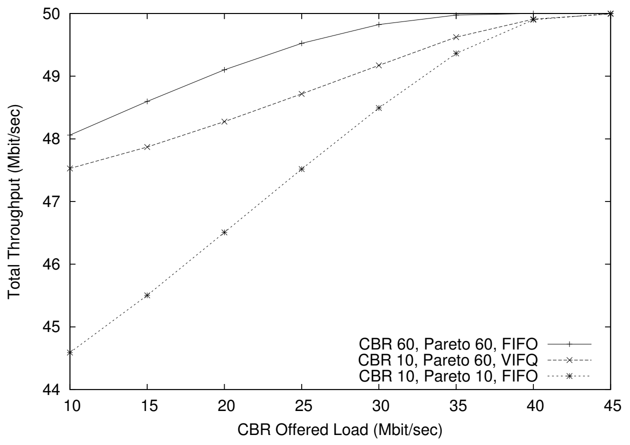

**Figure 10: Link Utilization - FIFO vs. VIFQ**

Figure 9 shows the average CBR throughput, depending on the offered load, with FIFO at 60 msec and VIFQ when using the 10 msec class. The results show the expected behaviour. There is a small penalty for choosing a lower-delay class, but throughput is close to the FIFO throughput.

Figure 10 shows the total throughput of CBR and Pareto traffic, depending on the fraction of CBR traffic in 3 scenarios: FIFO with a 10 msec and FIFO with a 60 msec buffer, compared to VIFQ with a 10- and a 60 msec class. Note that the y-axis begins at 44 Mbit/sec. It shows how the increased dropping of CBR traffic in the 10 msec class of VIFQ results in reduced link utilization compared to FIFO queueing. However, the results also show how a two-class 10/60 msec VIFQ system provides better link utilization than a single shared FIFO queue with a 10 msec buffer. Of course, less CBR and more Pareto traffic causes an overall higher impact of reduced buffer sizes. Overall, the utilization penalty of delay differentiation using VIFQ is rather small.

## 5.2 TCP vs. Pareto Sources

In another experiment, 5 service classes are configured with delay targets of 20, 40, 60, 80, and 100 msec. For each class, background traffic comprised of 32 Pareto sources with a total average offered load of 10% of the bottleneck capacity is created. The experiment studies the behaviour of TCP traffic comprised of 1, 10, 50, and 100 flows. Figure 11 shows the TCP throughput depending on the service class chosen for TCP. The results confirm the assumption that it is beneficial to pick a higher delay class, since TCP throughput generally benefits from more buffers, but the effect is diminished with an increasing number of flows.

A second similar experiment with a set of short TCP flows studies the effect of VIFQ on the completion times of short flows. The system is configured with the same 5 service classes and the same set of 32 Pareto background flows for each class, as before. A set of short flows is configured to start 20 flows/sec with each flow transmitting a 100KB file. The results are shown in Figure 12. As expected, flow completion times are reduced when choosing a higher delay class, since TCP throughput generally benefits from more buffers. In addition, the standard deviation (shown as error bars) does not change fundamentally between service classes. The average completion time for FIFO queueing is approximately 0.72 seconds and also shown in the figure as a benchmark. The standard deviation for FIFO queueing is not shown in the figure for clarity reasons, but it is about 0.61 seconds and thus fairly similar to the respective standard deviation for each VIFQ service class. This is evidence that using VIFQ, as well as the choice of service class, does not affect fairness more than FIFO queueing.

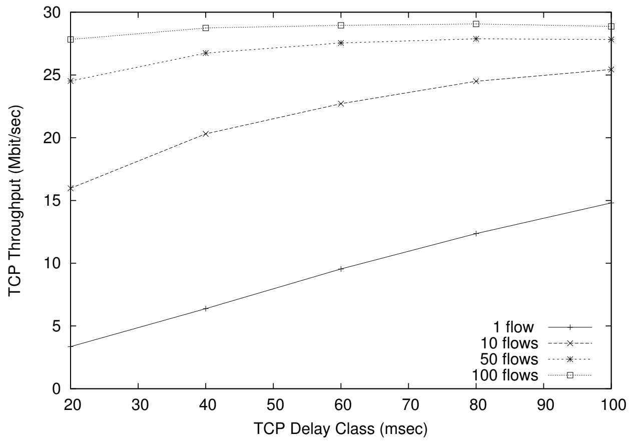

**Figure 11: TCP Throughput Depending on Delay Class**

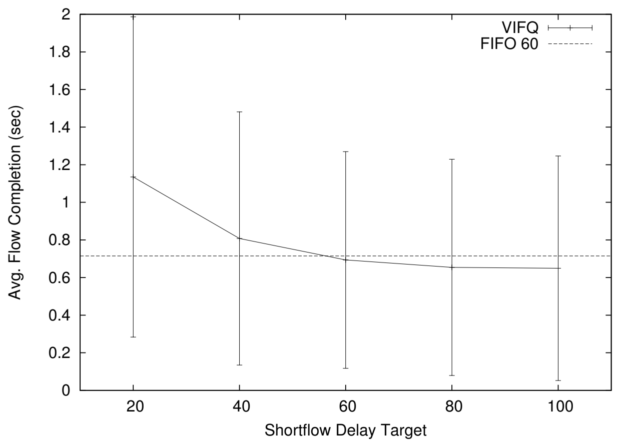

**Figure 12: Short Flows with Pareto Background**

## 5.3 TCP/TFRC with Different Delay Targets

The previous experiments demonstrate that VIFQ provides the expected incentives for feedback-controlled TCP traffic to pick the traffic class with more buffer space, if it competes with open-loop traffic. However, it is also interesting to study the impact that competing long-term TCP and TFRC flows have on each other to verify the basic assumptions about VIFQ. This is done by a series of experiments with two sets of TCP or TFRC flows in different service classes between 10 and 100 msec. Due to space restrictions, only the results for two sets of 50 TCP and TFRC flows each are shown in Figure 13 as an illustration. The results show that the TCP and TFRC rate control algorithms have a preference for low round-trip times over more buffer space, even with only 100 competing flows overall. This is not entirely surprising and confirms the observations that are reported in the small router buffer work described in Section 2.2.1.

## 5.4 Integrated Scenario

This experiment investigates a scenario with more traffic classes and a more comprehensive traffic mix. The scenario is comprised of 4 service classes with delay targets of 10, 20, 60, and 100 msec. 6 sets of flows use the available classes as shown in Table 2. It is assumed that a "rogue" set of Pareto sources chooses the 10 msec class (last row in Table 2). Figure 14 shows the resulting average throughput for each traffic set for FIFO (with 60 msec buffering) and VIFQ. The standard deviations are shown as error bars. The results illustrate that VIFQ provides the proper incentive to pick a service class. The "rogue" Pareto traffic in the 10 msec class suffers a penalty with VIFQ, while the long-term TCP flows are able to pick up the corresponding service allocation.

The delay plot for one run of these experiments is shown in Figure 15 to illustrate the delay differentiation capabilities of VIFQ. Note that the plot for CBR 10 traffic completely overlaps with the plot for the "rogue" Pareto traffic that transmits in the same service class. Also, the plots for both types of TCP flows in the 60 msec class overlap.

## 6. CONCLUSION

The goal of VIFQ is to manage queueing delays without any side effects on the "natural" (FIFO) throughput of traffic classes. VIFQ is a versatile building block for service differentiation, because it can provide multi-class delay differentiation without direct admission control or traffic management requirements. VIFQ can be implemented with low execution complexity and deployed with very low operational overhead. Further, VIFQ in isolation can provide some form of useful service differentiation without violating net neutrality principles. Although there is no comprehensive analytical model for VIFQ yet, the paper presents evidence for these claims in the form of simulation experiments. More analytical and experimental work is needed to fully investigate VIFQ. Another interesting area for future work is the combination of VIFQ with active queue management schemes.

While the presented VIFQ scheduling algorithm is fundamentally not more complex than FIFO queueing, more prototyping and ex-

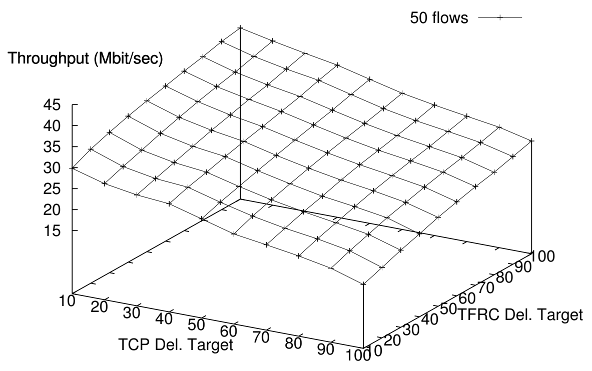

**Figure 13: Competing TCP and TFRC Flow Sets**

**Table 2: Traffic Mix for Integrated Scenario**

| Traffic | Service Class |
|---|---|
| 50 long-term TCP flows | 60 msec |
| Web TCP flows 10/sec, 100KB | 60 msec |
| CBR 20% load | 20 msec |
| CBR 20% load | 10 msec |
| Pareto, 32 sources, 20% load | 100 msec |
| Pareto, 32 sources, 20% load | 10 msec |

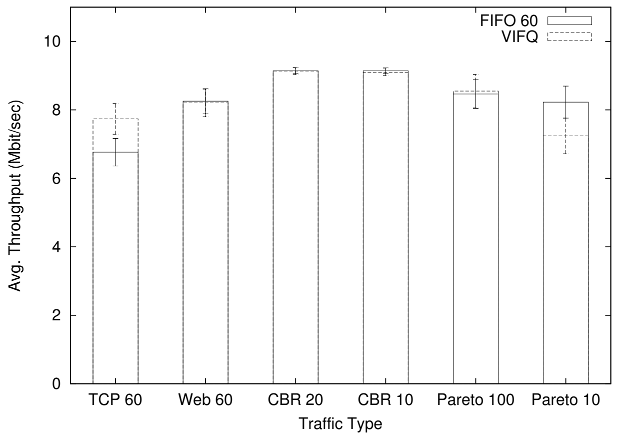

**Figure 14: Throughput for Multiple Classes**

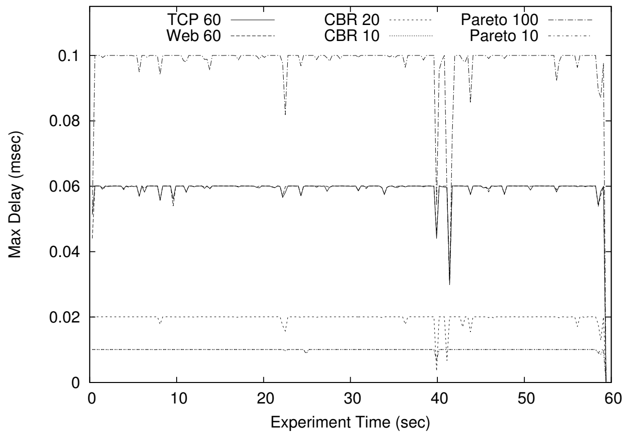

**Figure 15: Delay Differentiation for Multiple Classes**

perimental work is needed to fully assess the execution overhead of VIFQ, especially on practical hardware platforms. I.e., VIFQ requires managing more counters and slightly more involved queueing mechanisms than FIFO, so future work needs to establish that the actual execution overhead is comparable to FIFO and manageable at very high line rates with contemporary hardware.

Ultimately, more research is needed to fully explore the design space of rate-transparent scheduling and potential traffic mixes, especially when it comes to closed-loop traffic. If it is deemed important to maintain the status-quo of current TCP fairness, then TCP implementations will need to adhere to certain standard rules for choosing the service class. This is not an unusual requirement. For example, the notion of TCP fairness is also based on the implicit assumption that all end systems use a similar configuration of parameters, such as the initial window size or the slope of the window increase during congestion avoidance.

## Acknowledgements
Kate Larson formulated the original proof used in Section 4.2.

## 7. REFERENCES

[1] The Aggregate Magic Algorithms. http://aggregate.org/MAGIC. accessed Aug 12, 2011.

[2] The ns-2 Network Simulator. http://nsnam.isi.edu/nsnam. accessed Aug 12, 2011.

[3] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. *Proceedings of SIGCOMM 2004 - ACM CCR*, 34(4):281–292, 2004.

[4] S. Ayyorgun and R. L. Cruz. A Service-Curve Model with Loss and a Multiplexing Problem. In *Proceedings of ICDCS 2004*, pages 756–765, Mar. 2004.

[5] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, and G. Salmon. Experimental Study of Router Buffer Sizing. In *Proceedings of IMC 2008*, pages 197–210, Oct. 2008.

[6] J. C. R. Bennett, K. Benson, A. Charny, and J.-Y. L. William F. Courtney. Delay Jitter Bounds and Packet Scale Rate Guarantee for Expedited Forwarding. *IEEE/ACM Transactions on Networking*, 10(4):529–540, Aug. 2002.

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

[8] B. Briscoe, A. Jacquet, C. Di Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe. Policing Congestion Response in an Internetwork using Re-feedback. *Proceedings of SIGCOMM 2005 - ACM CCR*, 35(4):277–288, 2005.

[9] R. Bush and D. Meyer. RFC 3439 - Some Internet Architectural Guidelines and Philosophy, Dec. 2002.

[10] A. Charny and J. Y. L. Boudec. Delay Bounds in a Network with Aggregate Scheduling. In *Proceedings of QofIS 2000*, pages 1–13. Springer LNCS 1922, Sept. 2000.

[11] F. Ciucu, O. Hohlfeld, and L. Y. Chen. On the Convergence to Fairness in Overloaded FIFO Systems. In *Proceedings of INFOCOM 2011*, Apr. 2011.

[12] B. Davie, A. Charny, J. Bennett, K. Benson, J.-Y. L. W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. RFC 3246 - An Expedited Forwarding PHB (Per-Hop Behavior), Mar. 2002.

[13] A. Dhamdhere and C. Dovrolis. Open Issues in Router Buffer Sizing. *ACM Computer Communications Review*, 36(1):87–82, Jan. 2006.

[14] A. Dhamdhere, H. Jiang, and C. Dovrolis. Buffer Sizing for Congested Internet Links. In *Proceedings of INFOCOM 2005*, pages 1072–1083, Mar. 2005.

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

[16] P. Eardley. RFC 5559 - Pre-Congestion Notification (PCN) Architecture, June 2009.

[17] V. Firoiu, X. Zhang, and Y. Guo. Best Effort Differentiated Services: Trade-off Service Differentiation for Elastic Applications. In *Proceedings of ICT 2001*, June 2001.

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

[19] J. Gettys. Bufferbloat: Dark Buffers in the Internet. *IEEE Internet Computing*, 15(3):96–95, May 2011.

[20] Y. Ghiassi-Farrokhfal and J. Liebeherr. Output Characterization of Constant Bit Rate Traffic in FIFO

Networks. *IEEE Communication Letters*, 13(8):618–620, 2009.

[21] 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.

[22] P. Hurley, M. Kara, J.-Y. L. Boudec, and P. Thiran. A Novel Scheduler For a Low Delay Service Within Best-Effort. In *Proceedings of IWQoS'01*, pages 389–403. Springer LNCS 2092, June 2001.

[23] Intel Corp. Intel IXP2805 Network Processor - Programmer's Reference Manual, Apr. 2006. Order Number: 310015, Revision: 007US.

[24] Intel Corp. IntelÂő 64 and IA-32 Architectures Optimization Reference Manual, Dec. 2008. Order Number: 248966-017.

[25] N. Jin and S. Jordan. On the Feasibility of Dynamic Congestion-Based Pricing in Differentiated Services Networks. *IEEE/ACM Transactions on Networking*, 16(5):1001–1014, Oct. 2008.

[26] M. Karsten. Approximation of Generalized Processor Sharing with Stratified Interleaved Timer Wheels. *ACM/IEEE Transactions on Networking*, 18(3):708–721, June 2010.

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

[28] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. *Proceedings of SIGCOMM 2002 - ACM CCR*, 32(4):89–102, 2002.

[29] F. Kelly. Models for a self-managed Internet. In *Philosophical Transactions of the Royal Society A358*, pages 2335–2348, 2000.

[30] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts. Evaluating the Number of Active Flows in a Scheduler Realizing Fair Statistical Bandwidth Sharing. *Proceedings of SIGMETRICS 2005 - ACM PER*, 33(1):217–228, 2005.

[31] T. Lakshman, A. Neidhardt, and T. J. Ott. The Drop from Front Strategy in TCP and in TCP over ATM. In *Proceedings of INFOCOM 1996*, pages 1242–1250, Mar. 1996.

[32] A. Lakshmikantha, C. Beck, and R. Srikant. Impact of File Arrivals and Departures on Buffer Sizing in Core Routers. *IEEE/ACM Transactions on Networking*, 19(2):347–358, Apr. 2011.

[33] M. Mathis. Reflections on the TCP Macroscopic Model. *ACM Computer Communications Review*, 38(1):47–49, Jan. 2009.

[34] B. Melander, M. Bjorkman, and P. Gunningberg. First-Come-First-Served Packet Dispersion and Implications for TCP. In *Proceedings of IEEE Globecom 2002*, pages 2170–2174, Nov. 2002.

[35] 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 Transactions on Networking*, 2(2):137–150, Apr. 1994.

[36] M. Podlesny and S. Gorinsky. RD Network Services: Differentiation through Performance Incentives. *Proceedings of SIGCOMM 2008 - ACM CCR*, 38(4):255–266, 2008.

[37] R. S. Prasad, C. Dovrolis, and M. Thottan. Router Buffer Sizing for TCP Traffic and the Role of the Output/Input Capacity Ratio. *IEEE/ACM Transactions on Networking*, 17(5):1645–1658, Oct. 2009.

[38] K. Ramakrishnan, S. Floyd, and D. Black. RFC 3168 - The Addition of Explicit Congestion Notification (ECN) to IP, Sept. 2001.

[39] S. Ryu, C. Rump, and C. Qiao. Advances in Active Queue Management (AQM) Based TCP Congestion Control. *Telecommunication Systems*, 25(3-4).

[40] H. Sariowan, R. L. Cruz, and G. Polyzos. SCED: A Generalized Scheduling Policy for Guaranteed Quality of Service. *IEEE/ACM Transactions on Networking*, 7(5):669–684, Oct. 1999.

[41] J. B. Schmitt, F. A. Zdarsky, and M. Fidler. Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch... In *Proceedings of INFOCOM 2008*, pages 1669–1677. IEEE, Apr. 2008.

[42] G. Varghese and T. Lauck. Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility. In *Proceedings of SOSP 1987*, pages 25–38, New York, NY, USA, Nov. 1987. ACM.

[43] C. Villamizar and C. Song. High Performance TCP in ANSNET. *ACM SIGCOMM Computer Communications Review*, 24(5):45–60, Oct. 1994.

[44] M. Welzl and D. Ros. RFC 6297 - A Survey of Lower-than-Best-Effort Transport Protocols, June 2011.