Computer Networks 55 (2011) 2392–2406

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A practical fair queuing scheduler: Simplification through quantization

Z. Dwekat [a], G.N. Rouskas [b,*]

[a] Sprint Network Services, Raleigh, NC, USA
[b] Department of Computer Science, North Carolina State University, Engineering Building 2, 890 Oval Drive, Raleigh, NC 27695-8206, USA

ARTICLE INFO

Article history:
Received 9 July 2010
Received in revised form 4 January 2011
Accepted 8 April 2011
Available online 16 April 2011
Responsible Editor: A.K. Somani

Keywords:
Packet scheduling
Fair queuing
Tiered service

ABSTRACT

The design of fair packet schedulers involves a tradeoff between implementation complexity, on one hand, and delay and fairness guarantees, on the other. In this paper, we present tiered-service fair queuing (TSFQ), a new scheduler that exploits certain properties of Internet traffic to speed up the bottleneck operations related to virtual time computation and packet sorting. Specifically, TSFQ makes innovative use of quantization (along the two dimensions of flow weights and packet lengths) her with specialized yet simple queuing structures. TSFQ combines all three properties that are important to a fair queuing algorithm, namely, a tight delay bound, worst-case fairness, and low complexity and amenability to hardware implementation. Hence, we believe that, for network operators, deploying TSFQ scheduling has the potential to enhance their ability to offer and guarantee a wide range of services.

1. Introduction

The Internet has evolved into a ubiquitous global communication medium that carries a constantly evolving traffic mix that is becoming richer as innovation and technology improvements spawn new telecommunications applications and services. Specifically, real-time and interactive applications (including audio and video streaming, multimedia conferencing, etc.) require performance bounds from the network in terms of bandwidth, delay, or delay jitter. For instance a voice-over-IP (VoIP) application requires both a minimum bandwidth (generally, between 20 and 80 Kbps, depending on the voice codec used) and a round-trip delay of about 150 ms (dictated by human ergonomics) to ensure a "good" user experience. Hence, the network must support some form of quality of service (QoS) functionality in order to serve large numbers of heterogeneous users and applications with a wide range of requirements in terms of throughput and delay.

In packet-switched networks, packets from various users (flows) have to share the network resources, including buffer space at the routers and link bandwidth. Whenever resources are shared, contention arises among users seeking service. Consequently, shared resources employ a *scheduling discipline* to resolve contention by determining the order in which users receive service. In particular, the scheduling algorithm is a central component of the QoS architecture of packet-switched networks, and the performance that applications receive is directly affected by the service discipline employed by the nodes along their path.

Fig. 1 illustrates the general scheduler model that we will use in our discussion. Specifically, we assume that the scheduler serves n flows and employs per-flow queuing such that an arriving packet belonging to flow i, $i = 1, \ldots, n$, is inserted at the tail of the queue dedicated to this flow. As a result, each flow queue is sorted in increasing order of packet arrival times and its packets are served in a FCFS order. As shown in the figure, each flow i is associated with a positive real weight ϕ_i that is determined in advance (e.g., based on the application's bandwidth or delay requirements, or on economic factors). The scheduler uses the weights in some discipline-specific manner to determine which of the head-of-line packets in the flow queues to serve next.

[*] Corresponding author.
E-mail address: rouskas@csc.ncsu.edu (G.N. Rouskas).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.04.004

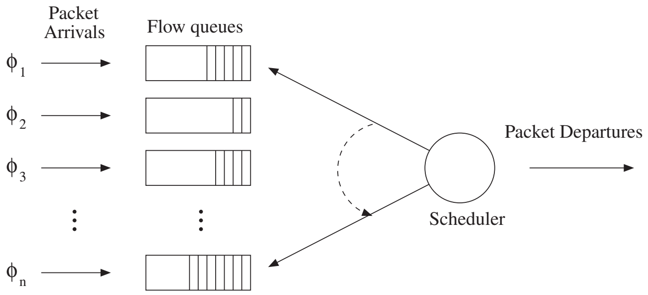

Fig. 1. Model of link scheduler serving n flows; ϕ_i is the weight assigned to flow *i*.

A packet scheduler is desirable to possess three important properties [11]: (1) isolation and fairness, (2) performance bounds, and (3) low algorithmic complexity. We elaborate on these properties next.

Isolation and fairness. The scheduler must provide isolation among the competing flows in order to ensure that each flow receives its allocated/fair share of the link bandwidth. Isolation prevents misbehaving flows (e.g., flows transmitting too fast) from affecting other flows sharing the same link. For best-effort applications, fairness typically refers to *max–min fair allocation* [3] of the link bandwidth among the flows, whereby flows with "small" bandwidth demands receive what they want while flows with "large" demands receive an equal share of the remaining link bandwidth. For applications with a guaranteed rate, two measures of fairness are typically considered:

- *Proportional fairness.* Proportional fairness, a measure introduced by Golestani [7], essentially requires that the difference between the normalized service received by any two backlogged flows *i* and *j*, over any time period (t_1,t_2), be bounded by a constant *C*:

$$\left|\frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j}\right| \leqslant C, \tag{1}$$

  where r_i is the allocated rate of flow *i* and $S_i(t_1,t_2)$ is the amount of data of flow *i* served during the time period (t_1,t_2).
- *Worst-case fairness.* Worst-case fairness, introduced by Bennett and Zhang [2], is a more refined notion of fairness. Rather than comparing the relative amounts of service received by two flows *i* and *j*, it compares the service received by a single flow *i* to the service it would receive in the ideal case, i.e., when *i* has exclusive access to an output link of rate r_i. Specifically, let D_i denote the maximum time a packet of flow *i* arriving to an empty queue will have to wait before receiving its guaranteed rate r_i, under a certain service discipline. D_i is called the *worst-case fair index (WFI)* of flow *i*. A service discipline is called *worst-case fair* if, for all flows *i*, D_i is bounded by a constant that depends on the discipline but is independent of the queues of the other flows served by the scheduler.

Performance bounds. Many streaming and multimedia applications require a throughput bound, i.e., they must be served at a minimum data rate. In addition, certain real-time and/or interactive applications may require bounds on packet delay. Such bounds may be expressed *deterministically* (e.g., in the form of a worst-case delay that no packet must exceed) or *statistically* (i.e., in the form of a delay threshold and a probability that any packet's delay will not exceed the threshold). Other performance bounds that have been considered include bounds on the delay jitter and on packet loss.

Low algorithmic complexity. A link scheduler may need to select the next packet to serve every time a packet departs. As optical link speeds increase from a few Gbps currently to tens of Gbps and beyond, a scheduler may have only a few microseconds or less to make a decision. Hence, in order to operate at wire speeds, the scheduling discipline must be amenable to hardware implementation and require few, preferably simple, operations. In particular, since the links of backbone networks may serve hundreds of thousands of flows simultaneously, the number of operations involved in making a scheduling decision should be independent of the number of flows sharing the link.

These requirements are often contradictory, and the design of packet schedulers involves a tradeoff between implementation complexity, on one hand, and fairness and performance guarantees, on the other. In general, schedulers can be classified according to their internal structure in one of two broad classes:

1. *Timestamp-based schedulers* maintain a global variable, usually referred to as *virtual time*, to sort arriving packets and serve them in that order. Timestamp schedulers have good delay and fairness properties but high complexity, hence they have found limited deployment in high-speed routers. One of the differentiating characteristics of timestamp scheduler variants is their approach to alleviating the complexity, as we discuss in the next section.
2. *Frame-based schedulers* divide time into slots of fixed or variable length, and assign slots to flows in some sort of round-robin fashion. Schedulers such as deficit round robin (DRR) [19] are easy to implement in hardware and have been widely deployed in commercial routers. However, frame-based schedulers have poor behavior in terms of delay bound and fairness.

In this paper, we present *tiered-service fair queuing (TSFQ)*, a new packet scheduler that exploits certain properties of Internet traffic to speed up the bottleneck operations related to virtual time computation and packet sorting: that applications typically require one of a small number of service levels [17,18], and that the Internet packet length distribution exhibits a small number of prominent modes. Specifically, TSFQ makes innovative use of quantization (along the two dimensions of flow weights and packet lengths) together with specialized yet simple queuing structures. TSFQ combines all three properties that are important to a fair queuing algorithm, namely, a tight delay bound, worst-case fairness, and low complexity and amenability to hardware implementation.

The remainder of this paper is organized as follows. In Section 2, we review the operation of timestamp schedulers and classify them according to their approach to reducing the implementation complexity. In Sections 3–5 we introduce the TSFQ scheduler and describe the details of its operation. We present an experimental evaluation of TSFQ in Section 6, and we conclude the paper in Section 7.

2. Timestamp-based schedulers

Timestamp schedulers emulate the ideal but unimplementable generalized processor sharing (GPS) algorithm [15]. GPS is an ideal scheduler, a theoretical construct that serves both as a starting point for designing practical scheduling disciplines and as a reference point for evaluating the fairness and delay properties of these disciplines. GPS visits each backlogged flow queue in turn and serves an infinitesimal fraction of the head-of-line packet at each queue. If flows are assigned different weights ϕ_i (refer to Fig. 1), then the service they receive from GPS is proportional to their weight.

If a queue is empty, GPS skips it to serve the next non-empty queue. Therefore, whenever some queues are empty, backlogged flows will receive additional service in proportion to their weights. Consequently, GPS achieves an *exact* max–min weighted fair bandwidth allocation [11]. It also provides isolation (protection) among flows, since a misbehaving flow is restricted to its fair share and does not affect other flows.

GPS is defined in a theoretical fluid flow model in which multiple queues may be served simultaneously. In a practical packet system, on the other hand, packet transmissions may not be preempted and only one queue may be served at any given time. The timestamp schedulers we review in the following subsections are designed for packetized systems, and emulate the operation of GPS by maintaining a virtual time function. Packets are assigned a timestamp based partly on the virtual time value at the time of their arrival, and are transmitted in increasing order of timestamp. In general, timestamp schedulers have good delay and fairness properties, but high implementation complexity. The complexity of such schedulers arises from two factors.

1. *Virtual time computation.* In order to assign a timestamp to an arriving packet, the scheduler must compute the virtual time function at the time of arrival; for n backlogged flows, this computation may take time $O(n)$.
2. *Packet sorting.* The scheduler selects among the head-of-line packets of backlogged flows the one with the smallest timestamp to serve next; this operation takes time $O(\log n)$ using a priority queue. The logarithmic complexity and the fact that the priority queue structure is not suited to hardware implementation pose significant challenges.

Most timestamp scheduler variants can be classified according to their approach to alleviating the complexity. Several schedulers use approximate virtual time functions that are more efficient to compute, reducing the $O(n)$ worst-case complexity to as low as $O(1)$. Other schedulers employ some form of *quantization* (e.g., in virtual time, flow weight, or both) to reduce the complexity of the packet sorting operations. In the following subsections, we review the operation and properties of representative schedulers from each category, including some that employ both techniques. For the sake of completeness, as well as to introduce notation that we will use later for describing the TSFQ scheduler, we start our discussion with two schedulers that do not use any simplification.

2.1. Emulating GPS: no simplification

The first two schedulers we review do not attempt to simplify the virtual time computation or packet sorting operations. They differ in the scheduling policy they use, which has implications on their respective fairness properties.

2.1.1. Weighted fair queuing (WFQ)

Weighted fair queuing (WFQ) [6,15] is an approximation of GPS that serves packets in the order they would complete service had they been served by GPS. Therefore, the WFQ scheduler needs to emulate the operation of the GPS server. To this end, a virtual time function $V(t)$ was proposed in [15] to track the progress of GPS. The rate of change of $V(t)$ is:

$$\frac{\vartheta V(t+\tau)}{\vartheta \tau} = \frac{1}{\sum_{i\in B(t)}\phi_i} \tag{2}$$

where $B(t)$ denotes the set of backlogged flows at time t and ϕ_i is the weight assigned to flow i. Let r be the rate of the link (server). In GPS, if flow i is backlogged at time t, it receives a rate of

$$\frac{\vartheta V(t+\tau)}{\vartheta \tau}\phi_i r = \frac{\phi_i}{\sum_{i\in B(t)}\phi_i} r. \tag{3}$$

In other words, $V(t)$ is the marginal rate at which backlogged flows receive service in GPS.

Suppose that the kth packet of flow i arrives at time a_i^k, and has length L_i^k. Let S_i^k and F_i^k denote the virtual times at which this packet begins and completes service, respectively, under GPS. Letting $F_i^0 = 0$ for all flows i, we have [15]:

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\} \tag{4}$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i}. \tag{5}$$

The WFQ scheduler serves packets in increasing order of their virtual finish times F_i^k, a policy referred to as "smallest virtual finish time first (SFF)" [1].

Let us consider the complexity of WFQ. At packet departure instants, the SFF policy is used to select the next packet to transmit. This selection can take $O(\log n)$ time, where n is the number of (backlogged) flows, if packet virtual finish times are organized in a heap-based priority queue data structure. In addition, there is the cost of maintaining the virtual time function $V(t)$ at packet arrival and departure instants. The worst-case complexity of computing $V(t)$

can be $O(n)$, although the average-case complexity is $O(1)$ [7]. Therefore, WFQ is expensive to implement within core routers that may handle hundreds of thousands to millions of flows at any given time.

The degree to which WFQ approximates GPS is determined by two properties that were established in [15]:

- *Bounded delay property*. A packet will finish service in a WFQ system no later than the time it would finish in the corresponding GPS system plus the transmission time of a maximum size packet.
- *Weak service property*. The service (in terms of total number of bits) that a flow receives in a WFQ system does not fall behind the service it would receive in the fluid GPS system by more than one maximum packet size.

While due to the second property above a WFQ system may not fall behind GPS by more than one maximum packet size, it may in fact be ahead of GPS in terms of the service provided to some flows. In particular, it was shown in [2] that WFQ may introduce substantial unfairness relative to GPS in terms of the worst-case fairness index (WFI). Specifically, GPS has a WFI of zero, but the WFI of WFQ increases linearly with the number of flows n. Consequently, there may be substantial discrepancies in the service experienced by individual flows under the WFQ and GPS schedulers.

2.2. Worst-case fair weighted fair queuing (WF^{2}Q)

The WF^{2}Q algorithm was introduced in [2] as a better packet approximation of GPS than WFQ. Specifically, WF^{2}Q employs a "smallest eligible virtual finish time first (SEFF)" policy for scheduling packets. A packet is *eligible* if its virtual start time is no greater than the current virtual time; hence, the WF^{2}Q scheduler only considers the packets that have started service in GPS to select the one to be transmitted next. It has been shown [2] that WF^{2}Q is work-conserving, maintains the bounded delay property of WFQ, and has these additional two properties:

- *Strong service property*. The service (in terms of total number of bits) that a flow receives from a WF^{2}Q system cannot fall behind (respectively, be ahead of) the service it would receive in the fluid GPS system by more than one maximum packet size (respectively, a fraction of the maximum packet size).
- *Worst-case fairness property*. WF^{2}Q is worst-case fair, i.e., its WFI is a constant independent of the number n of flows served by the scheduler.

The first property implies that the WF^{2}Q scheduler closely tracks the GPS system in terms of the service received by each flow, and due to the second property, WF^{2}Q is an optimal packet scheduler in terms of worst-case fairness [2]. However, the worst-case complexity of WF^{2}Q is $O(n)$, identical to that of WFQ, as both schedulers need to compute the virtual time function $V(t)$.

2.3. Emulating GPS: simplification through efficient virtual time functions

2.3.1. WF^{2}Q+

A lower-complexity scheduler, WF^{2}Q+ was introduced in [1]. The WF^{2}Q+ scheduler is work-conserving, has the same bounded delay, strong service, and worst-case fairness properties of WF^{2}Q, but uses a different virtual time function that can be computed more efficiently than the function $V(t)$ in (2) used by WFQ and WF^{2}Q. The new function is [1]:

$$V_{WF^2Q+}(t+\tau) = \max\left\{V_{WF^2Q+}(t)+\tau, \min_{i\in B(t)}\left\{S_i^{h_i(t)}\right\}\right\}. \tag{6}$$

In the above expression, $B(t)$ is the set of backlogged flows at time t, $h_i(t)$ is the sequence number of the packet at the head of flow i's queue at time t, and $S_i^{h_i(t)}$ is the virtual start time of that packet. The minimum operation in the right-hand side of (6) can be performed in time $O(\log n)$ in the worst-case using a priority queue structure, hence the overall complexity of WF^{2}Q+ is $O(\log n)$, significantly lower than the $O(n)$ complexity of WFQ and WF^{2}Q.

As pointed out in [1], the WF^{2}Q+ scheduler implementation can be further simplified by maintaining a single pair of start and finish virtual time values per flow, rather than on a per-packet basis. Specifically, only a single pair of values, S_i and F_i, needs to be maintained for each flow i, corresponding to the virtual start and finish times, respectively, of the packet at the head of the queue of flow i. Let $Q_i(t-)$ denote the queue size of flow i just before time t. When a new packet reaches the head of the queue at time t, the values of S_i and F_i are updated according to the following expressions [1]:

$$S_i = \begin{cases} F_i, & Q_i(t-)\neq 0, \\ \max\{F_i, V_{WF^2Q+}(t)\}, & Q_i(t-)=0, \end{cases} \tag{7}$$

$$F_i = S_i + \frac{L_i^k}{\phi_i}, \tag{8}$$

where L_i^k is the length of this packet and ϕ_i is the weight assigned to the flow.

Overall, the WF^{2}Q+ scheduler achieves tight delay bounds and good worst-case fairness with a relatively low $O(\log n)$ algorithmic complexity.

2.3.2. Self-clocked fair queuing (SCFQ)

The $O(n)$ worst-case algorithmic complexity of the WFQ and WF^{2}Q schedulers is due to the fact that the order of packet transmissions in these queuing schemes is determined by tracking the progress of the fluid-flow GPS reference system, which, in turn, requires the computation of the virtual time function $V(t)$ whose rate of change is defined in (2). Self-clocked fair queuing (SCFQ) [7] avoids the computationally expensive emulation of a hypothetical reference system by adopting a self-contained approach to fair queuing. Specifically, instead of using a virtual time to compute the finish times of packets as in expressions (4) and (5), SCFQ computes the finish time F_i^k of the kth packet of flow i as:

$$F_i^k = \max\{F_i^{k-1}, F_{cur}\} + \frac{L_i^k}{\phi_i}, \tag{9}$$

where F_{cur} is the finish time of the packet currently in service, and finish times are initialized to $F_i^0 = 0$ for all flows i. Since the finish times can be computed in $O(1)$ time using expression (9), the algorithmic complexity of SCFQ is $O(\log n)$ because of the requirement to select the packet with the smallest finish time for transmission.

Although the rule (9) that SCFQ uses to compute packet finish times is easy to implement, the tradeoff is a much larger delay bound than WFQ. In particular, the delay bound provided by SCFQ increases linearly with the number n of flows served by the scheduler, in the worst case [8]. The worst-case fair index (WFI) of SCFQ is the same as that of WFQ, i.e., proportional to the number n of flows.

2.3.3. Start-time fair queuing (SFQ)

Start-time fair queuing (SFQ) [9] is a variant of SCFQ that maintains both a start time and a finish time for each packet. Upon arrival, the kth packet of flow i is assigned the start time:

$$S_i^k = \max\{F_i^{k-1}, S_{cur}\}, \tag{10}$$

where S_{cur} is the start time of the packet in service at the time of arrival. The finish time F_i^k of the kth packet is computed as:

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i}. \tag{11}$$

Unlike the other packet fair schedulers we have considered so far, SFQ serves packets in increasing order of their *start times*, not their finish times.

It can be seen that expressions (10) and (11) may be computed in constant time, hence SFQ has the same low algorithmic complexity $O(\log n)$ as SCFQ. However, it has been shown [9] that the worst-case delay of SFQ is significantly lower than with SCFQ. The worst-case fairness properties of SFQ are similar to those of WFQ and SCFQ.

2.4. Emulating GPS: simplification through quantization

Even with simplified virtual time functions, timestamp-based schedulers incur a substantial per-packet overhead that is related to selecting the packet with the smallest timestamp to be transmitted next. The schedulers we discuss in this section use some form of quantization to reduce the complexity of packet sorting operations.

2.4.1. Bin sort fair queuing (BSFQ)

Bin sort fair queuing (BSFQ) [5] uses quantization in the (virtual) time domain in a way that reduces the computational effort required for sorting packets. To this end, virtual time is divided into slots (bins) of length Δ, where Δ is a configurable parameter, and the scheduler maintains a virtual system clock that is equal to the left endpoint of the current slot. Arriving packets are assigned a virtual finish time using an expression similar to the one for SCFQ in (9), that can be computed in constant time. Packets with finish times that fall within the same slot, are inserted in a first-in, first-out (FIFO) queue associated with this slot. In other words, there is no sorting of packets that have finish times “close” to each other, as determined by the length Δ of a slot. Therefore, this “bin sorting” operation takes $O(1)$ time. When the virtual clock is equal to the left endpoint of slot i, the scheduler serves all the packets in the FIFO queue associated with slot i. When all the packets of the queue have been transmitted, the virtual clock is incremented by Δ and the scheduler serves the FIFO queue of the next slot $i + 1$.

The BSFQ scheduler is scalable and is easy to implement in hardware. Its fairness and delay guarantees depend strongly on the value of parameter Δ. When Δ is large, BSFQ reduces to FCFS, while when Δ is small, its operation is similar to that of SCFQ. While smaller Δ values result in better fairness and delay guarantees, the amount of state information that the scheduler needs to maintain increases and its efficiency decreases as the value of Δ decreases. Therefore, determining an appropriate value for Δ is a complex task that involves several tradeoffs.

2.4.2. Stratified round-robin (SRR)

Stratified round-robin (SRR) [16] also uses quantization, but in the domain of flow weights. Specifically, flows are grouped (“stratified”) into *flow classes* based on their weights. An exponential grouping is used, such that the kth flow class consists of flows i with weights such that: $\frac{1}{2^k} \leqslant \phi_i < \frac{1}{2^{k-1}}$. SRR has two scheduling components: an *intra-class* scheduler and an *inter-class* scheduler. The inter-class scheduler assigns a *scheduling interval* to each flow class such that the kth class is assigned an interval of length 2^k slots. Within a class, flows are scheduled in the associated scheduling intervals using a variant of DRR [19] that gives each flow a quantum that is proportional to its weight.

SRR has low complexity and provides delay and fairness guarantees similar to those of DRR. However, it improves the worst-case delay a single packet may experience to a small constant, whereas under DRR and BSFQ this value is proportional to the number n of flows served by the scheduler.

2.4.3. Fair round-robin (FRR)

The fair round-robin (FRR) scheduler [25] is similar to SRR in that flows are grouped into classes using the same exponential grouping, and has the same structure in that it employs both an intra-class and an inter-class scheduling component. The inter-class scheduler is timestamp-based, and determines the time a packet from each class is to be scheduled by taking into account the time-varying weight of each class (which changes over time as flows within a class become active or inactive). FRR assigns finish times to *flow classes*, not individual flows, by keeping track of the corresponding GPS system. This scheduler always serves the *eligible* flow class with the smallest finish time; eligibility of a flow class is defined as a generalization of the eligibility criterion introduced by the WF^{2}Q scheduler. Since the inter-class scheduler operates on the basis of flow classes, emulating GPS (i.e., computing the virtual time function) takes time proportional to the number m of classes, which, for a given system is a small constant (determined by the exponential grouping employed) that is independent of the number n of flows.

The intra-class scheduler has two functions. First, it needs to compute the class weight to pass to the inter-class scheduler; the latter uses these weights to determine the order in which each class is served, as we explained above. Second, it must decide the order in which packets from the various flows within the class will be transmitted whenever the inter-class scheduler serves this class. The intra-class scheduler uses a frame-based approach similar to DRR, but with a modification to account for the weight differences among the flows within the same class.

The FRR scheduler has low algorithmic complexity, is worst-case fair, and provides a constant delay bound, similar to SRR.

3. Tiered-service fair queuing (TSFQ)

In this section we introduce a new scheduler that is designed to achieve the delay and fairness properties of WF^{2}Q+ at low algorithmic complexity. As we discussed in Section 2.3.1, WF^{2}Q+ closely emulates the ideal GPS fluid-flow scheduler, and thus it combines tight delay bounds with a constant worst-case fair index. However, its complexity is $O(\log n)$, where n is the number of flows served. Although other scheduler variants have lower complexity, they provide looser delay and fairness guarantees than WF^{2}Q+. As a result, the service received by individual flows under these simpler schedulers may be significantly different than the service they would receive under GPS.

We now note that the schedulers we reviewed in the previous section were designed under the assumption that both flow weights and packet sizes may take *arbitrary* values. In other words, they are designed to allocate the link bandwidth in the finest possible granularity; taken to the limit, such schedulers could potentially allocate bandwidth at increments of 1 bit per second. Clearly, the ability to support arbitrary rates offers maximum flexibility in utilizing the available network capacity.

On the other hand, supporting rate allocation at such extremely fine granularity may seriously complicate the operation and management of the network. For instance, the task of differentiating between two rates that differ by, say, even a few Kbps, may be difficult or even impossible to accomplish for traffic of finite duration, undermining the network's ability to support important functions such as robust traffic policing or accurate customer billing. Furthermore, given the unpredictability of future bandwidth demands in terms of their size, arrival time, and duration, link capacity across a continuous-rate network may become fragmented. Such fragmentation poses significant challenges in terms of traffic engineering, and may compromise the ability to achieve an acceptable level of utilization or meet users' QoS requirements. With respect to packet schedulers, the assumption of arbitrary flow weights is a fundamental one in that it underlies the high complexity of the virtual time computation and packet sorting operations. As a result, the implementation of such schedulers suffers from severe scalability challenges that have impeded their adoption in the Internet.

Our work is motivated by two important observations regarding Internet traffic characteristics which suggest that the implementation of packet schedulers may be simplified significantly without compromising their delay and fairness properties.

- *Flow weights.* First, traffic flows are unlikely to have arbitrary weights. For instance, flows of guaranteed-service applications may be grouped into a small set of classes depending on the nature of the application (e.g., "voice", "video," "game,", etc.) with the flows in each class having similar bandwidth and delay requirements. Similarly, whereas best-effort applications have elastic requirements that adjust to the available rate, their rate is typically limited by the access bandwidth available to the user. Since most Internet service providers offer some type of tiered service [17], users may select only from a small set of bandwidth tiers. The practical implication of this fact is that the rates requested by flows (equivalently, the flow weights in the fair queuing system) are not arbitrary, but are limited to a small set of values that are typically known in advance. As we explain shortly, it is possible to speed up considerably the computation of the virtual time function if the scheduler is designed so as to handle only a small set of discrete flow weights.
- *Packet sizes.* The second observation is that in the Internet, the vast majority (i.e., up to 90%) of packets have a fixed length that takes one of a small number of values [23,20]. Therefore, the scheduler may employ appropriate queuing structures that simplify, or even completely eliminate the need for, packet sorting operations.

The main contribution of our work is a new scheduler that exploits the above observations by employing quantization in two dimensions: along the domain of flow weights and along the domain of packet sizes. We refer to this scheduler as *tiered-service fair queuing (TSFQ)*. As we will show later, TSFQ performs the two main scheduling operations, namely, computing the virtual time function and selecting the next packet to be transmitted, in time that is independent of the number n of flows, yet it achieves the excellent delay and fairness properties of the WF^{2}Q+ scheduler.

We consider a link scheduler which serves n flows and employs per-flow queuing, i.e., it allocates a FIFO buffer to each flow, as shown in Fig. 1. The scheduler supports p distinct tiers of service, where $p \ll n$ is a small constant (e.g., $p \approx 10 - 15$). The lth tier is characterized by a positive real weight ϕ_l, $l = 1,\ldots,p$. Each flow i is mapped to one of the p service levels, i.e., it is assigned one of the p weights ϕ_l; we assume that this assignment remains fixed throughout the duration of the flow. The mapping of flows to service tiers is performed at the time the flow enters the network by taking into account the QoS requirements of the application or the bandwidth tier to which the user subscribes. We make the assumption that the link is configured with the number p of service tiers and the associated weights ϕ_l. These parameters may be determined in advance by the network provider as part of the network planning process, by using empirical information regarding the user demands [13,18].

Before proceeding, we emphasize that the quantization of flow weights in TSFQ is different from the approach taken by the SRR [16] and FRR [25] schedulers we discussed above. Specifically, SRR and FRR allow flows to have *arbitrary* weights, but "stratify" them into a small number of classes using exponential grouping. In contrast, TSFQ assigns the *same* weight to *all flows* within a service tier. To make the distinction clear, we use the term *flow tier* to refer to a set of flows with the same weight, instead of the term *flow class* that was used in [16,25] to refer to a group of flows with similar weights as determined by the specific exponential grouping method employed.

3.1. Logical operation: flow weight quantization

The TSFQ scheduler operates in a manner similar to WF^{2}Q+ in that:

- it uses the same virtual time function shown in expression (6);
- it maintains a single pair of values, S_i and F_i for each flow i, corresponding to the virtual start and finish times, respectively, of the packet at the head of the FIFO queue of flow i; these values are updated according to expressions (7) and (8); and
- it employs the SEFF policy to serve packets.

The TSFQ scheduler logically consists of two components, as illustrated in Fig. 2, that together effect the quantization in the domain of flow weights. The first component comprises of p identical *intra-tier schedulers*, while the second component is a single *inter-tier scheduler*. The main function of each component is as follows:

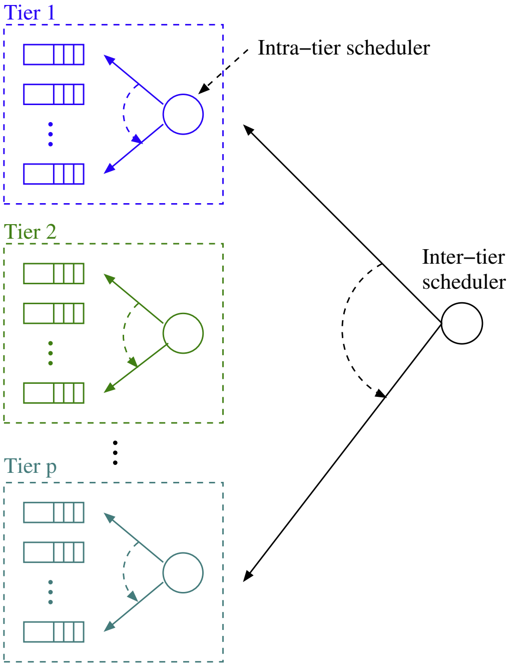

Fig. 2. Logical diagram of the TSFQ scheduler with p service tiers.

- *Intra-tier scheduler.* The lth intra-tier scheduler uses the SEFF policy to select, *among the flows of the lth service tier, $l = 1, \ldots, p$, only*, the flow i with the minimum virtual finish time F_i. The structure and operation of the intra-tier scheduler are described in detail in the following sections.
- *Inter-tier scheduler.* The inter-tier scheduler simply serves the packet at the head of the queue with the smallest virtual finish time among the p flows selected by the corresponding intra-tier schedulers. Since p is a small constant for the given link, the packet to be transmitted next can be determined in time that is independent of the number of flows, and in fact, this operation can be performed in constant time in hardware. Hence, the implementation of the inter-tier scheduler is straightforward and does not require any priority queue data structure to be maintained.

We note that the logical structure of the TSFQ scheduler is similar to the structure of the SRR and FRR schedulers which both consist of *intra-class* and *inter-class* scheduling components. However, there are significant differences in the functionality and operation of these schedulers. Specifically, the inter-class scheduler of SRR assigns scheduling intervals to each flow class, while the inter-class scheduler of FRR assigns weights to flow classes, not individual flows, and serves the class with the smallest timestamp. The TSFQ inter-tier scheduler, on the other hand, simply serves the flow with the smallest finish time among the p such flows across the p tiers, hence its operation is much simpler. The intra-class scheduler of SRR serves flows within its assigned scheduling interval using a variant of DRR, while the intra-class scheduler of FRR also uses a (different) variant of DRR. In contrast, the intra-tier scheduler of TSFQ (described shortly) uses simple queuing structures to maintain the flows within its tier sorted in decreasing order of finish time.

3.2. Virtual time computation

Let $S^{(l)}(t)$, $l = 1, \ldots, p$, denote the virtual start time of the flow with the smallest finish time among the flows of the lth service tier at time t. Then, we may rewrite the expression (6) of the virtual time function as:

$$V_{TSFQ}(t+\tau) = \max\left\{V_{TSFQ}(t) + \tau, \min_{l=1,\ldots,p}\left\{S^{(l)}(t)\right\}\right\}. \qquad (12)$$

Assuming that at any time t each intra-tier scheduler keeps track of the flow within its tier with the smallest finish time, the minimum operation in the right-hand side of expression (12) can be implemented in $O(1)$ time. Hence, the virtual time computation takes time that is independent of the number n of flows.

So far, we have shown that, due to the quantization of flow weights, both the virtual time computation and the inter-tier scheduling operations take time that depends only on the number p of tiers, which is a small constant for a given scheduler. Therefore, the critical component of TSFQ is the intra-tier scheduler which is responsible for identifying (selecting) the flow with the minimum virtual finish among the flows in its tier. In the next two sections we show that by employing simple queuing structures this selection operation can be implemented efficiently.

4. Intra-tier scheduler: the fixed-size packet case

The lth TSFQ intra-tier scheduler, $l = 1,\ldots,p$, serves flows belonging to the lth service tier and have been assigned the same weight ϕ_l. The p intra-tier schedulers are identical and operate independently of each other. Therefore, in this and the next section we consider the operation of a single intra-tier scheduler in which all flows have identical weights. For simplicity, we let ϕ denote the weight assigned to all the flows served by the scheduler.

In this section we make the additional assumption that *all packets of all flows have constant size L* (i.e., $L_i^k = L \forall i, k$). We will remove this assumption in the next section; however, we note that the implementation we present in this section is of practical importance to ATM networks. We also note that this case of limited flow rates (weights) and fixed packet sizes has been considered in [21]. Nevertheless, the queue structure and operation we present here is different as it applies specifically to the SEFF policy.

The following operation is based on the fact that in a system with fixed-size packets and flows of identical weight, sorting flows according to their virtual start times produces an identical order to sorting them according to their virtual finish times [21]. This property is formally expressed in the following lemma.

Lemma 4.1. *Consider flows i and j with $\phi_i = \phi_j = \phi$ and packets of fixed size L. Let S_i be the virtual start time of flow i, and S_j be the virtual start time of flow j. Then:*

$$S_i \leqslant S_j \Longleftrightarrow F_i \leqslant F_j \qquad (13)$$

4.1. Queue structure and operation

The intra-tier scheduler for fixed-length packets consists of a simple FIFO queue, as illustrated in Fig. 3. The scheduler maintains a single *token* κ_i for each flow i that it serves. Initially (i.e., at time $t = 0$, before any packet arrivals to the system), the FIFO queue is empty. Tokens are inserted at the tail of the FIFO queue representing the order in which flows will be served, and a token is removed from the head of the FIFO queue whenever it is selected for service by the inter-tier scheduler.

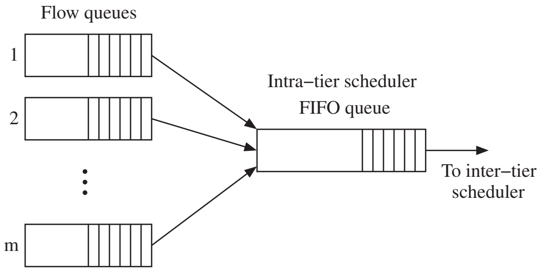

Fig. 3. Queue structure of the intra-tier scheduler for fixed-size packets.

The operation of the scheduler is fully described by the actions taken whenever a relevant event takes place. The relevant events occur when (1) a packet arrives, (2) a flow becomes eligible for service, or (3) a packet departs (is served).

- *Packet arrival.* A packet of flow i arriving at time t is inserted at the tail of this flow's queue. If flow i was active just prior to the arrival t (i.e., its queue was non-empty, hence $Q_i(t-) \neq 0$, using the notation of Section 2.3.1), then no other action is taken. If, on the other hand, flow i was inactive prior to the arrival (i.e., $Q_i(t-) = 0$), then this arriving packet reaches the head of this flow's queue at time t, and the start time S_i and finish time F_i of flow i are updated according to expressions (7) and (8), respectively. If this previously inactive flow i becomes eligible at time t (i.e., $S_i \leqslant V_{TSFQ}(t)$ after the update), then the next event is triggered, otherwise no other action is taken.
- *A flow becomes eligible for service.* When a flow i becomes eligible at time t (i.e., $S_i = V_{TSFQ}(t)$), then the token κ_i corresponding to this flow is inserted at the tail of the scheduler's FIFO queue.
- *Packet departure.* Let κ_i be the token at the head of the scheduler's FIFO queue at the time the inter-tier scheduler selects this tier to serve. Then, the packet at the head of the queue of flow i is served and token κ_i is removed from the scheduler's FIFO queue. If flow i becomes inactive, then no other action is taken. Otherwise, a new packet reaches the head of this flow's queue, and the start time S_i and finish time F_i are updated according to expressions (7) and (8), respectively. If the flow becomes eligible, then the corresponding event above is triggered, otherwise no action is taken.

Based on these actions, it is easy to see that token κ_i is in the scheduler's FIFO queue *if and only if* flow i is eligible for service. Therefore, we have the following results.

Lemma 4.2. *Considering only the flows of a given tier, the intra-tier scheduler of Fig. 3 is identical to the WF^{2}Q+ scheduler [1].*

Proof. Since tokens are inserted into the FIFO queue at the moment the corresponding flows become eligible for service (i.e., at the moment their virtual start time becomes equal to the current time), tokens in the FIFO queue are sorted in increasing order of the corresponding flows' virtual start times. Because of Lemma 4.1, the queue is sorted in increasing order of the virtual finish times, which is the order in which flows are served under WF^{2}Q+. Since (1) token arrivals to the FIFO queue take place at exactly the same instants that the corresponding head-of-line packets are considered for service under WF^{2}Q+, and (2) the order of service is identical, the two schedulers are identical under the assumption of flows with fixed-size packets and identical weights. □

Lemma 4.3. *The TSFQ scheduler consisting of p intra-tier schedulers and one inter-tier scheduler is identical to WF^{2}Q+.*

Proof. Each of the p intra-tier schedulers maintains a FIFO queue that sorts the flows in its tier in increasing order of their start (equivalently, finish) times, identical to the order in which they are considered under WF^{2}Q+. The inter-tier scheduler serves the p flows with tokens at the head of the p intra-tier FIFO queues in increasing order of their virtual start (finish) times. Consequently, the TSFQ scheduler overall is identical to WF^{2}Q+. □

Based on these results and the discussion in Section 3.2, we conclude that the TSFQ (intra- and inter-tier) scheduler achieves the worst-case fairness and delay properties of WF^{2}Q+ with an algorithmic complexity of $O(1)$. Note that this conclusion does not contradict the findings of [24] which suggest that the $O(\log n)$ time complexity is fundamental to achieving good delay bounds. The analysis in [24] assumes that flow weights and packet sizes can take arbitrary values, whereas the result of Lemma 4.3 only holds under the specific assumptions of fixed flow weights and packet lengths.

5. Intra-tier scheduler: the variable-size packet case

We now remove the assumption we made in the previous section that all packets have a fixed size. As in the previous section, we consider the problem of scheduling flows within a given service tier, therefore we assume that all flows are assigned the same weight ϕ. In a network with variable-size packets, the statement of Lemma 4.1 is no longer true, since the second term in the right-hand side of (8) is not constant. Hence, in such a network, fair queuing schedulers in general require some form of packet sorting.

In the Internet, however, it is well known that certain packet sizes dominate [23,20]. Specifically, the study in [23] found that packets of one of three common sizes make up more than 90% of all Internet traffic; the three common packet sizes identified in the study were 40, 576, and 1500 bytes, corresponding to TCP acknowledgments, the default IP datagram size, and maximum-size Ethernet frames, respectively. A more recent study [20] shows that (1) Internet traffic is mostly bimodal at 40 and 1500 bytes, (2) there is a shift away from 576 bytes due to the proliferation of Ethernet, and (3) a new mode is forming around 1300 bytes which the authors theorize is due to widespread use of VPNs. Similar studies, which can be found on CAIDA's web site (http://www.caida.org), confirm that the length of the vast majority of Internet packets takes one of a small number of constant values.

In the remainder of this section we show how these facts regarding the Internet packet length distribution naturally lead to simple queuing structures that exploit quantization along the domain of packet lengths. This modified version of the intra-tier TSFQ scheduler we presented in the previous section may handle Internet traffic efficiently in terms of packet sorting operations.

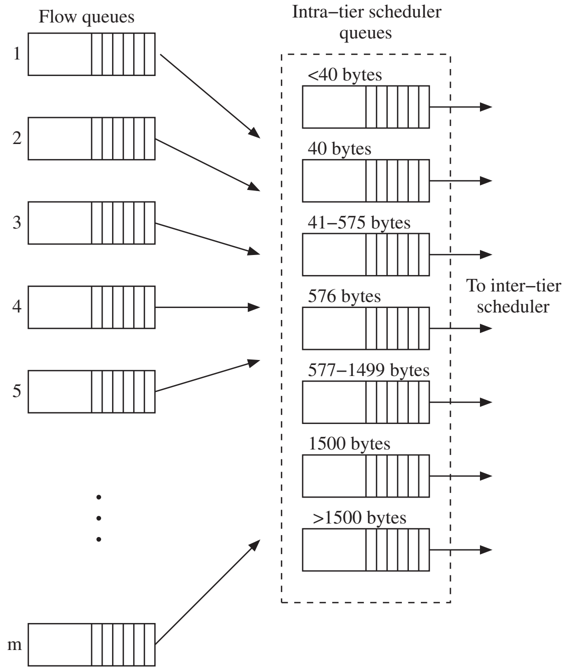

Fig. 4. Queue structure of the intra-tier scheduler for Internet packet traffic.

5.1. Queue structure and operation: packet size quantization

Instead of maintaining a single FIFO queue, as is the case for fixed-size packets shown in Fig. 3, the intra-tier scheduler for variable packet size networks maintains a small number k of queues. The queue structure of this scheduler is illustrated in Fig. 4 for the trimodal packet length distribution reported in [23]; the queue structure can be modified in a straightforward manner to reflect any similar distribution. In this case, the scheduler maintains $k = 7$ queues. Three of the queues are dedicated to packets of a common size, i.e., 40, 576, and 1500 bytes, respectively, which define the three modes of the distribution in [23]. The other four queues are for packets of size other the common values; as seen in Fig. 4, there is one queue for packets of size less than 40 bytes, one for packets of size 41–575 bytes, one for packets of size 577–1499 bytes, and one for packets of size greater than 1500 bytes.

The operation of the intra-tier scheduler is very similar to the one we described in Section 4.1, with only one difference. In particular, the actions taken at packet arrival and departure events are identical to those in the fixed-packet case listed in Section 4.1. The only difference is in the actions taken at instants when a flow becomes eligible for service:

- *A flow becomes eligible for service.* When a flow i becomes eligible at time t, then the token κ_i associated with this flow is inserted into the queue corresponding to the size of the packet at the head of the queue of flow i.

Similar to the fixed-packet case, token κ_i is in one of the intra-tier scheduler's queues if and only if flow i is eligible for service.

Since each of the p inter-tier schedulers maintains k distinct queues, the inter-tier scheduler selects the flow to serve next as the one with the smallest virtual finish time among the pk candidate flows whose tokens are at the head of the pk queues. Since both p and k are small integers and their values are constant for a given system, this operation of the inter-tier scheduler takes constant time, as in the fixed-size packet case.

5.2. Packet sorting operations

Note that Lemma 4.1 holds true for packets of a common size. Hence, the queues dedicated to these packets operate in a FIFO manner, and packets are simply inserted at the tail of these queues. Since packets of a common size make up more than 90% of Internet traffic [23], no sorting operations are necessary for the large majority of packets. On the other hand, queues dedicated to packets of size between the common values must be sorted appropriately at the time of a packet insertion. These sorting operations take place infrequently (e.g., about 10% of the time), and involve relatively short queues (since about 10% of the packets are spread over several such queues at p different service levels). Moreover, the time complexity of the sorting operations is independent of the number m of flows in the given service tier, and is a function only of the network load and the ratio of packets with a non-common size.

We have the following results.

Lemma 5.1. *The TSFQ scheduler for variable packet sizes, consisting of p intra-tier schedulers as in Fig. 4 and one inter-tier scheduler, is identical to WF^{2}Q+.*

Proof. The proof of Lemmas 4.2 and 4.3 also holds in this case, hence the scheduler is equivalent to WF^{2}Q+. □

Finally, we note that, although consecutive packets of the same flow i may be inserted into different queues in Fig. 4, they will always be transmitted in order: not only does the second packet have a larger virtual finish time than the first one, but since there is exactly one token for each flow, the second packet cannot be considered for service until the first one has departed from the scheduler.

5.3. Elimination of packet sorting operations

The operation of the intra-tier scheduler may be further simplified by eliminating packet sorting even for queues holding packets of size between the common values. Doing so may cause some packets to be served in incorrect order of virtual finish time, hence introducing a small degree of unfairness. However, the overall impact is likely to be small. Indeed, observe that packets of a non-common size represent only a small fraction of the overall traffic seen by the server, and are distributed over a number of different queues across p service tiers. Consequently, the arrival rate to each of these queues is likely to be low, especially under typical operating conditions when the load offered to the server is not too high. Now note that, since all flows within a service tier have the same weight ϕ in expression (8), the order of packets in such a queue will depend on the relative values of their virtual start time and length. Therefore, even when a small packet arrives to find larger packets in the queue (i.e., packets with a larger value for the second term in the right-hand side of (8)), the elapsed time since the previous arrival (which affects the first term of (8)) may be sufficiently large so that the queue remain sorted.

This intuition is further supported by the coarse manner in which the leap forward virtual clock [22] algorithm computes timestamps, and the mechanism employed by the bin sort fair queuing (BSFQ) discipline [5] to sort packets. The results in [22,5] indicate that approximate sorting can be as good as exact sorting; moreover, in the case of our TSFQ scheduler, approximate sorting is limited to a small fraction of all packets. This intuition is confirmed by the following lemma that bounds the delay of any packet in a queue corresponding to packets of sizes between two common sizes; note that the lemma suggests that the delay is relatively small, and can be further improved by reducing the range of packet sizes that are accommodated by any given queue.

Lemma 5.2. *Consider an intra-tier scheduler for a tier with rate r_l and a queue that is dedicated to packets of length between L_{lo} and L_{hi} bits, $L_{lo} < L_{hi}$. The maximum delay that a packet may experience due to the elimination of packet sorting operations in this queue is bounded by $r(L_{hi} - L_{lo})/r_l$, where r is the rate of the scheduler (over all tiers).*

Proof. Consider a scheduler operating at rate r with p service tiers and rate r_l assigned to tier l, $l = 1,\ldots,p$. Consider the lth tier, and let A denote the head-of-line packet of some flow that at time t is released to the queue of its intra-tier scheduler that corresponds to packets with sizes between L_{lo} and L_{hi} bits, $L_{lo} < L_{hi}$. Further assume that packets B_i, $i = 1,\ldots,m$, are already in that queue and no sorting operations are performed (i.e., the queue operates in a FIFO manner). In the worst case: (1) all packets B_i have finish times greater than that of packet A; and (2) packet A has length L_{lo} (so as to receive the minimum possible finish time).

Since packet A was released, i.e., it became eligible for service, at time t, we have that its start time $S_A = V(t)$. Packets B_i were also released before time t. Letting S_{max} denote the maximum start time of any of these packets, we have that:

$$S_{max} < V(t). \tag{14}$$

By assumption, the finish times of all packets B_i are greater than the finish time $F_A = S_A + L_{lo}/r_l = V(t) + L_{lo}/r_l$ of packet A. Letting F_{min} denote the minimum finish time of the B_i packets, we have: $F_{min} > V(t) + L_{lo}/r_l$. Since the packets B_i have length no larger than L_{hi}, we also have that

$$S_{min} \geqslant F_{min} - \frac{L_{hi}}{r_l} > V(t) + \frac{L_{lo}}{r_l} - \frac{L_{hi}}{r_l}. \tag{15}$$

From (14) and (15) we can bound the difference between the minimum and maximum start times of packets B_i:

$$S_{max} - S_{min} = \frac{L_{hi}}{r_l} - \frac{L_{lo}}{r_l}. \tag{16}$$

Since the minimum and maximum start times correspond to the virtual times $V(t_{min})$ and $V(t_{max})$, respectively, that the corresponding packets B_i were released to the queue, we also have that:

$$V(t_{max}) - V(t_{min}) = \frac{L_{hi}}{r_l} - \frac{L_{lo}}{r_l} \Rightarrow t_{max} - t_{min} \leqslant \frac{L_{hi}}{r_l} - \frac{L_{lo}}{r_l}, \qquad (17)$$

where the inequality in the second line of (17) is due to the fact that $V(t_2) - V(t_1) \geqslant t_2 - t_1$ for any $t_2 > t_1$.

Assuming that the server is not oversubscribed, i.e., the sum of the rates assigned to all flows is no larger than r, we can now write for the m packets B_i:

$$\sum_{B_i, i=1,\cdots,m} \frac{L_i}{r_l} \leqslant r(t_{max} - t_{min}) = \frac{r(L_{hi} - L_{lo})}{r_l}. \qquad (18)$$

In other words, the total size of the packets B_i that were incorrectly inserted before packet A in the queue does not exceed $r(L_{hi} - L_{lo})$ bits, hence the maximum delay that the elimination of sorting operations will cause for packet A is bounded by $r(L_{hi} - L_{lo})/r_l$. □

Finally, we emphasize that the queue structure shown in Fig. 4 is for illustration purposes only and is simply meant to convey the idea underlying the structure of the scheduler for Internet packet traffic; we do not imply that routers have to be configured in exactly this manner. Network operators may configure this queue structure to reflect the specific packet distribution observed in their networks, and update it over time as traffic conditions evolve. Similarly, they may optimize the number of service tiers and the flow weights associated with them by taking into account the prevailing user demands [18,13]. Therefore, this framework of fair queuing schedulers for tiered-service networks is quite flexible. Network providers may adapt the specific elements of the framework to differentiate their offerings, and to provide users with a menu of customized services.

6. Experimental evaluation of TSFQ

We have developed implementations of the TSFQ scheduler for the *ns-2* network simulator and in the Linux kernel. The details of the *ns-2* implementation are reported in [4], along with a comprehensive set of simulation experiments that validate the operation of TSFQ. In this section we present network experiments with the Linux kernel implementation which is fully described in [12].

The TSFQ scheduler was implemented as a Linux kernel loadable module. The Linux kernel version 2.6.26.2 [14] was used, the latest kernel available at the time of the implementation. The WF^{2}Q+ discipline [1] was also implemented as a separate loadable module for comparison purposes, since a Linux kernel implementation of the WF^{2}Q+ scheduler did not exist at the time.

6.1. Testbed and experimental setup

The experiments were carried out using a testbed consisting of three Linux machines connected as shown in Fig. 5. The leftmost machine acts as the "sender" of UDP traffic that is destined to the rightmost machine, the "receiver." The middle machine is configured as a "router" that receives packet traffic from the sender and forwards it to the receiver. The Ethernet link from the sender to the router is configured to run at 1 Gbps, while the link from the router to the receiver is configured to run at 10 Mbps. Consequently, the latter link becomes the bottleneck, causing the queues at the router to build up.

UDP traffic at the sender is generated by multiple simultaneous flows, each transmitting to a different destination port on the receiver. The router implements the TSFQ and WF^{2}Q+ disciplines to schedule packets received from the sender for transmission on the outgoing 10 Mbps link. It also employs per-flow queuing, assigning a separate FIFO queue to each UDP flow. The router uses the port information carried by the packets to determine the flow to which they belong and insert them into the appropriate queue. The TSFQ and WF^{2}Q+ schedulers use pre-configured weights to serve the queues of the various flows.

The UDP flows at the sender continuously transmit packets to the receiver without any form of flow control. Packet sizes L are randomly generated from the following discrete distribution:

$$Pr[L=x] = \begin{cases} 0.3, & x=40, \\ 0.3, & x=1200, \\ 0.3, & x=1500, \\ 0.1, & 1 \leqslant x \leqslant 39,\ 41 \leqslant x \leqslant 1199,\ 1201 \leqslant x \leqslant 1499. \end{cases} \qquad (19)$$

This distribution generates traffic dominated by a small number (in this case, three) of packet sizes, and is similar to the packet size distributions observed in [23,20]. Consequently, the intra-tier schedulers of TSFQ are configured with six queues, similar to the structure shown in Fig. 4 (the seventh queue of Fig. 4 for packets of size greater than 1500 bytes is not used here, as no such packets are generated).

A number of experiments were carried out to investigate the behavior of the schedulers under three scenarios:

- *Scenario I.* Several flows of different weights are started at the same time. After the system reaches steady state, the flows are terminated one by one. This scenario explores how the schedulers allocate excess bandwidth to the remaining flows.
- *Scenario II.* A small number of flows are started at the same time. After the system reaches steady state, new flows of different weights are started. Once the system reaches steady state again, the newly introduced flows are terminated. The start and termination instants of

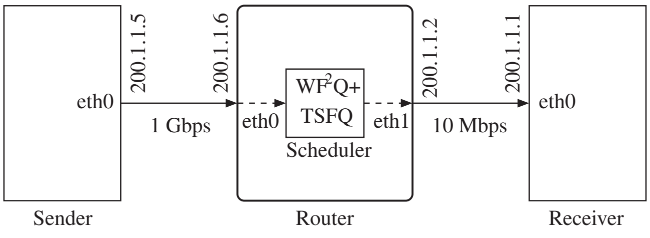

Fig. 5. Testbed setup.

the new flows are spread over time. These experiments are used to investigate the impact of new flows on the bandwidth share of existing ones, as well as the allocation of excess bandwidth.

- *Scenario III.* Many flows spanning a small number of service levels are run for a long time. This scenario is used to evaluate the fairness of each scheduling discipline. Specifically, we use Jain's fairness index [10] to compare the WF^{2}Q+ and TSFQ schedulers. In a system with n competing flows and flow i having throughput share f_i, $i = 1,\dots,n$, Jain's fairness index (FI) is defined as:

$$FI = \frac{\left(\sum_{i=1}^{n} f_i\right)^2}{n\sum_{i=1}^{n} f_i^2}, \tag{20}$$

such that a value of 1 represents perfect fairness with all flows receiving an equal share (=1/n) of the available bandwidth.

6.2. Performance results

In this section we present a set of illustrative experiments for the three different scenarios described above; a comprehensive suite of experimental results are available in [12].

6.2.1. Scenario I: Allocation of Excess Bandwidth

Figs. 6 and 7 present the results of an experiment to investigate the relative behavior of the WF^{2}Q+ and TSFQ schedulers in allocating excess bandwidth. This experiment involves four flows: flows 1 and 2 each have weight 0.15, while flows 3 and 4 each are assigned weight 0.35. For this experiment, the TSFQ scheduler was configured with $p = 2$ tiers, one with weight 0.15 and the other with weight 0.35; hence, flows 1 and 2 were assigned to the first tier, and flows 3 and 4 were assigned to the second tier. All four flows start transmission simultaneously at time $t = 0$, and are terminated one-by-one, in reverse order of their index, at 10 s intervals.

Figs. 6 and 7 plot the throughput of each flow (in Mbps) as a function of time for the WF^{2}Q+ and TSFQ schedulers, respectively. Recall that the bottleneck link in the experimental setup was set to 10 Mbps, and this latter value represents the bandwidth that is shared among the four flows. We observe that during the first 10 s of the experiment when all four flows are active, both schedulers allocate the available bandwidth in proportion to the flow weights, such that flows 1 and 2 (respectively, flows 3 and 4) capture approximately 15% (respectively, 35%) of the total bandwidth each. When flow 4 terminates, the bandwidth share of each of the three flows that remain active increases proportionally to its weight. In particular, flow 3 with the highest weight (0.35) captures most of the bandwidth that becomes available, while flows 1 and 2 of the same but lower weight (0.15) capture an equal share of the excess bandwidth. The same behavior is observed at the time the other flows are terminated. Importantly, the throughput curves of a given flow are comparable across the two figures, implying that the TSFQ and WF^{2}Q+ schedulers perform similarly in terms of allocating bandwidth to flows in proportion to their weights.

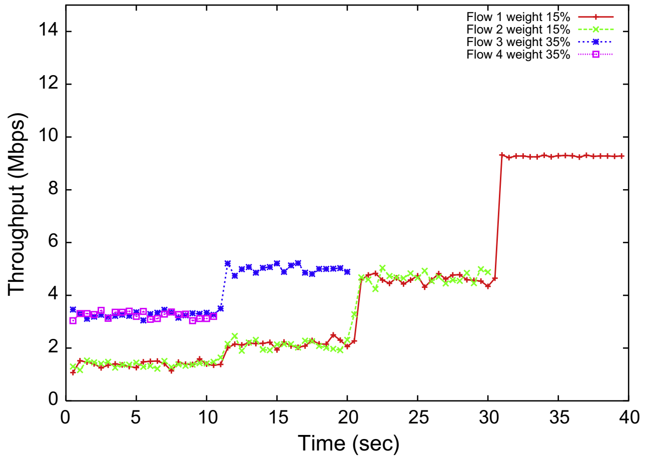

Fig. 6. *Scenario I*, four flows, WF^{2}Q+ scheduler.

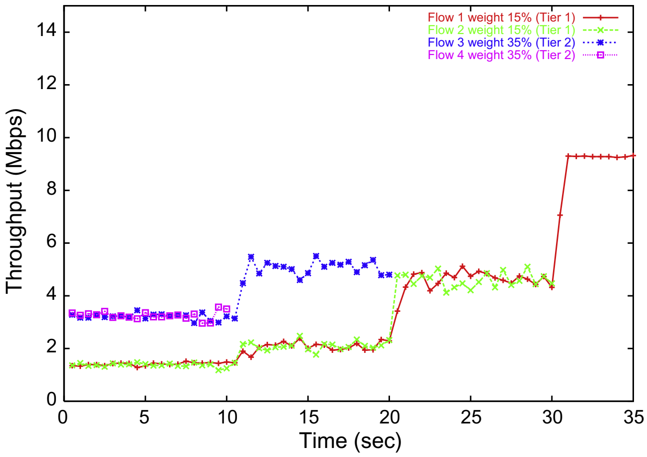

Fig. 7. *Scenario I*, four flows, TSFQ scheduler.

6.2.2. Scenario II: impact of new flows

In order to demonstrate the performance of the two schedulers when flows both arrive and depart, we run an experiment with the same flows as in Scenario I, i.e., two flows of weight 0.35 and two of weight 0.15. In this case, the flows of lower weight (flows 1 and 2) both become active at time $t = 0$. At time $t = 10$ s (respectively, $t = 20$ s) flow 3 (respectively, flow 4) of higher weight becomes active. All four flows remain active until time $t = 30$ s, at which time flow 3 departs, followed by flow 4 at time $t = 40$ s.

The results of this experiment are shown in Figs. 8 and 9, which again plot the time-varying throughput of each flow under the WF^{2}Q+ and TSFQ scheduler, respectively. During the first ten seconds of the experiment, the two active flows receive an equal share of the available bandwidth despite their low weights, as expected. As the other two flows are introduced, the bandwidth share of existing flows is reduced accordingly; on the other hand, the bandwidth share of active flows increases as flows depart (i.e., are terminated). Overall, we make three important observations: (1) at any point in time the available bandwidth is shared among active flows in proportion to

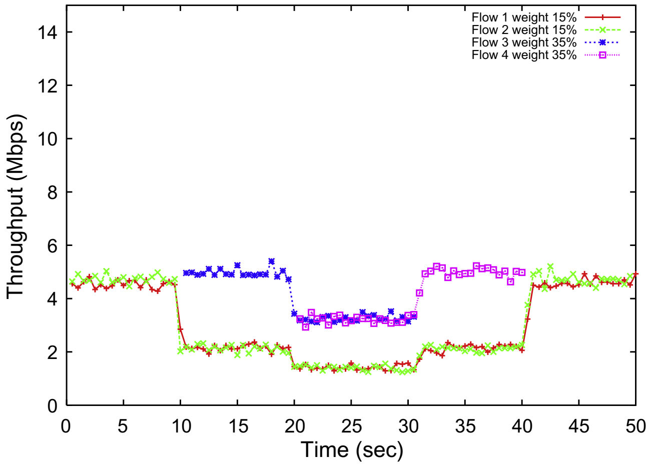

Fig. 8. *Scenario II*, four flows, WF^{2}Q+ scheduler.

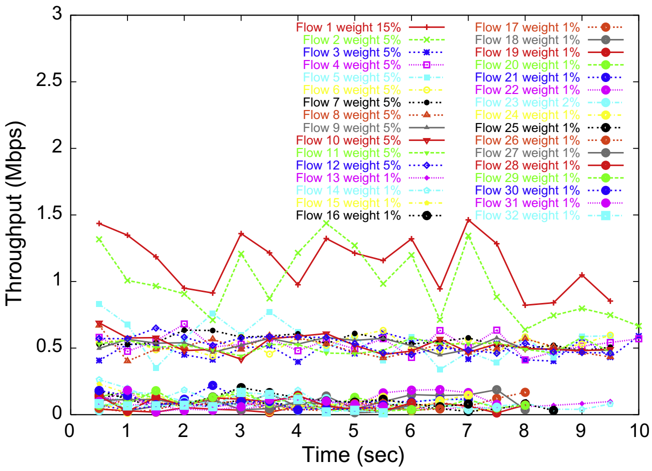

Fig. 11. *Scenario III*, thirty-two flows, TSFQ scheduler.

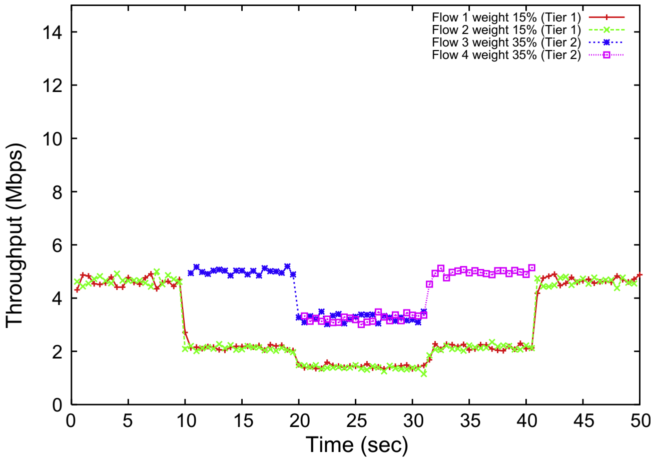

Fig. 9. *Scenario II*, four flows, TSFQ scheduler.

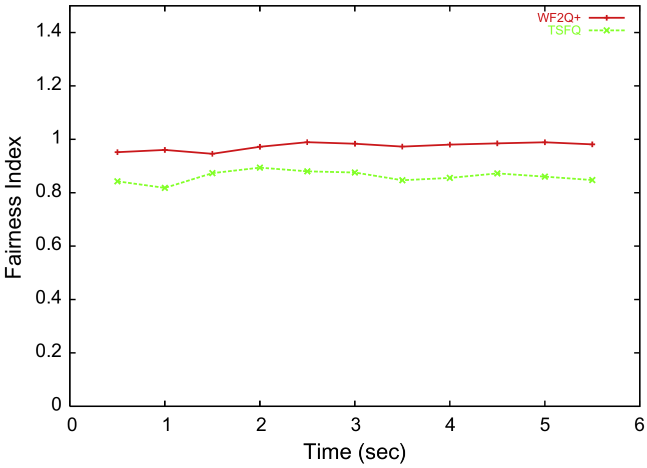

Fig. 12. *Scenario III*, thirty-two flows, fairness index.

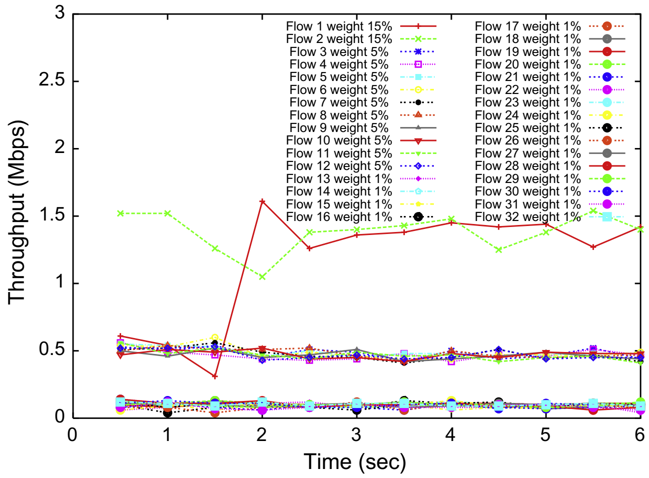

Fig. 10. *Scenario III*, thirty-two flows, WF^{2}Q+ scheduler.

their weights; (2) as flows arrive or depart, the bandwidth share of all the flows in the system quickly reaches a new equilibrium; and (3) there is good agreement in the behavior of the two schedulers.

6.2.3. Scenario III: long-term fairness

The first experiment of this section investigates qualitatively (i.e., graphically) the long-term fairness of the WF^{2}Q+ and TSFQ schedulers, and involves 32 flows that all start at time $t = 0$ and remain active throughout the duration of the experiment. Two of the flows have weight of 0.35, ten flows have weight equal to 0.05, and the remaining twenty flows have weight of 0.01. Hence, for this experiment, the TSFQ scheduler was configured with $p = 3$ tiers with weights of 0.35, 0.05, and 0.01, respectively, and the thirty-two flows were assigned to the appropriate tier according to their individual weights.

Figs. 10 and 11 plot the throughput of the thirty-two flows as a function of time for the WF^{2}Q+ and TSFQ schedulers, respectively. In both figures, the flows are clearly separated in three groups, each corresponding to three TSFQ tiers, with flows within each group receiving a share of bandwidth in line with their weight. Although the throughput of the various flows shows more short-term variations under the TSFQ scheduler, the overall behavior is similar in the two figures. In order to quantify the long-term fairness of the two schedulers, we computed Jain's fairness index from expression (20), using the long-term throughput of the thirty-two flows, and normalizing these values by the corresponding flow weight. The fairness index values are plotted in Fig. 12 as a function of time. The fairness index is about 10% higher under the WF^{2}Q+ scheduler, reflecting the lower throughput variations in Fig. 10. Nevertheless, the curves of both schedulers are relatively stable across the duration of the experiment, indicating that the two schedulers have similar fairness characteristics.

7. Concluding remarks

We have presented tiered-service fair queuing (TSFQ), a new scheduler motivated by two key observations: that applications typically require one of a small number of service levels, and that the Internet packet length distribution exhibits a small number of prominent modes. Within each tier, the schedulers employ a fixed number of queues to handle packets with few or no sorting operations. The intra-tier scheduler simply serves the packet with the smallest timestamp among a constant number of packets at the front of the intra-tier queues. The simple structure and operation of the schedulers are practically realizable and especially attractive for hardware implementation. The TSFQ scheduler is equivalent to WF^{2}Q+ with the additional property that the virtual time function can be computed in constant time. Therefore, we believe that employing TSFQ scheduling within high-speed routers will enable network operators to enhance significantly their ability to offer and guarantee a wide range of services.

Acknowledgments

The authors thank Ajay Babu Amudala Bhasker and Shrikrishna Khare for implementing the *ns-2* and Linux kernel modules, respectively, of TSFQ and WF^{2}Q+ as part of their MS theses.

References

[1] J.C.R. Bennett, H. Zhang, Hierarchical packet fair queueing algorithms, in: Proceedings of ACM SIGCOMM '96, 1996, pp. 143–156.

[2] J.C.R. Bennett, H. Zhang, WF^{2}Q: worst-case fair weighted fair queueing, in: Proceedings of IEEE INFOCOM '96, 1996, pp. 120–128.

[3] D. Bertsekas, R. Gallager, Data Networks, Prentice Hall, Inc, Englewood Cliffs, NJ, 1992.

[4] Ajay Babu, Amudala Bhasker, Tiered-service fair queueing (TSFQ): a practical and efficient fair queueing algorithm. Master's thesis, North Carolina State University, Raleigh, NC 2006.

[5] S. Cheung, C. Pencea, BSFQ: bin sort fair queueing, in: Proceedings of IEEE INFOCOM '02, 2002.

[6] A. Demers, S. Keshav, S. Shenker, Analysis and simulation of a fair queueing algorithm, in: Proceedings of ACM SIGCOMM '89, September 1989, pp. 1–12.

[7] S. Golestani, A self-clocked fair queueing scheme for broadband applications, in: Proceedings of IEEE INFOCOM '94, 1994, pp. 636–646.

[8] P. Goyal, H.M. Vin, Generalized guaranteed rate scheduling algorithms: a framework, IEEE/ACM Transactions on Networking 5 (4) (1997) 561–571.

[9] P. Goyal, H.M. Vin, H. Cheng, Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks, in: Proceedings of ACM SIGCOMM '96, August 1996, pp. 157–168.

[10] R. Jain, W. Hawe, D.M. Chiu, A quantitative measure of fairness and discrimination for resource allocation in shared systems, Technical Report TR-301, DEC Research Report, 1984.

[11] S. Keshav, An Engineering Approach to Computer Networking, Addison-Wesley, Reading, MA, 1997.

[12] Shrikrishna Khare, Testbed implementation and performance evaluation of the tiered service fair queueing (TSFQ) packet scheduling discipline, Master's thesis, North Carolina State University, Raleigh, NC, August 2008.

[13] Q. Lv, G.N. Rouskas, An economic model for pricing tiered-service networks, Annals of Telecommunications 65 (3–4) (2010) 147–161.

[14] Linux Kernel Organization. The Linux kernel archives. <http://www.kernel.org/>.

[15] A.K. Parekh, R.G. Gallager, A generalized processor sharing approach to flow control in integrated services networks: the single-node case, IEEE/ACM Transactions on Networking 1 (3) (1993) 344–357.

[16] S. Ramabhadran, J. Pasquale, Stratified round robin: a low complexity packet scheduler with bandwidth fairness and bounded delay, in: Proceedings of ACM SIGCOMM '03, August 2003, pp. 239–249.

[17] G.N. Rouskas, Internet Tiered Services: Theory, Economics, and Quality of Service, Springer, New York, 2009.

[18] G.N. Rouskas, N. Baradwaj, On bandwidth tiered service, IEEE/ACM Transactions on Networking 17 (6) (2009).

[19] M. Shreedhar, G. Varghese, Efficient fair queueing using deficit round robin, in: Proceedings of ACM SIGCOMM '95, 1995.

[20] R. Sinha, C. Papadopoulos, J. Heidemann, Internet packet size distributions: Some observations. <http://netweb.usc.edu/~rsinha/pkt-sizes/>, October 2005.

[21] D.C. Stephens, J.C.R. Bennett, H. Zhang, Implementing scheduling algorithms in high-speed networks, IEEE Journal on Selected Areas in Communications 17 (6) (1999) 1145–1157.

[22] S. Suri, G. Varghese, G. Chandranmenon, Leap forward virtual clock: an O(loglogN) queueing scheme with guaranteed delays and throughput fairness, in: Proceedings of IEEE INFOCOM '97, 1997.

[23] K. Thompson, G.J. Miller, R. Wilder, Wide-area internet traffic patterns and characteristics, IEEE Network 11 (6) (1997) 10–23.

[24] J. Xu, R. Lipton, On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms, in: Proceedings of ACM SIGCOMM '02, 2002.

[25] X. Yuan, Z. Duan, FRR: a proportional and worst-case fair round-robin scheduler, in: Proceedings of IEEE INFOCOM '05, March 2005.

Zyad Dwekat received B.S. in Electrical Engineering from Mutah University, Karak, Jordan in 1992, and M.S. in Computer Networking and Ph.D. degree in Electrical Engineering from the department of Electrical Engineering in North Carolina State University, Raleigh, NC in 2002 and 2009, respectively.

He has been working as a Network Engineer with Sprint/Nextel since 2002. Before that, he worked as Electrical Engineer with RJAF, Jordan from 1992 until 1999. He also interned at Alcatel Research Center Dallas, TX in 2001. He received Alcatel 2001 inventor award and filed five patents during his career with Sprint. He also works part time as Adjunct Professor at North Carolina Wesleyan College and Wake Tech, Raleigh, NC since 2009.

George Rouskas is a Professor of Computer Science at North Carolina State University. He received the Ph.D and MS degrees in Computer Science from the College of Computing, Georgia Institute of Technology, and an undergraduate degree in Computer Engineering from the National Technical University of Athens (NTUA), Athens, Greece. His research interests are in network design and optimization, network architecture, and performance evaluation. He is the author of a book on "Internet Tiered Services" (Springer, 2009), co-editor of the book "Traffic Grooming for Optical Networks" (Springer, 2008), and co-editor of the upcoming book "Next-Generation Internet Architectures and Protocols" (Cambridge University Press, 2010). He is a recipient of a 1997 NSF AREER Award, the 2004 ALCOA Foundation Engineering Research Achievement Award and the 2003 NCSU Alumni Outstanding Research Award, and he was inducted in the NCSU Academy of Outstanding Teachers in 2004. He is the founding editor-in-chief of the Optical Switching and Networking journal, and is serving as the co-chair of the Optical Networks and Systems Symposium for IEEE Globecom 2010 and as co-chair of the IEEE ICCCN 2011 conference. He has served as associate editor for IEEE/ACM Transactions on Networking and the Computer Networks journal, and he has organized several networking conferences.