SI-WF^{2}Q: WF^{2}Q Approximation with Small Constant Execution Overhead - Extended Version

Martin Karsten
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 6B2, Canada
Email: mkarsten@cs.uwaterloo.ca
Technical Report CS-2006-01, Version: January 6, 2006 / 16:09

***Abstract*— This paper presents a novel priority queue data structure and access operations for timer maintenance in the context of traffic shaping and scheduling in packet networks. The data structure and operations are used to construct an efficient General Processor Sharing (GPS) approximation scheduler. In contrast to existing proposals, this scheduler has constant and near-optimal delay and fairness properties, *and* can be implemented with bounded amortized O(1) algorithmic complexity, *and* has very low absolute execution overhead. The paper presents the data structure, the scheduling algorithm, and studies its execution complexity. The scheduling properties are analyzed and shown to relate nicely to existing work. Some illustrative simulation results are presented to reaffirm those properties.**

***Index Terms*— Communication systems, Computer network performance, Scheduling, Data structures, Algorithms**

I. Introduction

Packet scheduling algorithms are a cornerstone for the future development of packet-switched networks as ubiquitous communication infrastructure, integrating a wide range of network technologies and offering a wide variety of application services. Packet scheduling algorithms increase the level of control over packet transmission and permit the support of different service policies. *General Processor Sharing* (GPS) has been introduced in [1] as a conceptual packet scheduler with many desirable properties. In very basic terms, GPS scheduling works by assigning fractions of the overall forwarding capacity to flows. These fractions are termed *weights* and the GPS scheduler guarantees fluid service in proportion to a flow's weight compared to the sum of all other active flows' weights. A large variety of GPS emulation algorithms have been proposed, but so far no algorithm exists that combines very close GPS approximation with constant algorithmic complexity *and* low execution overhead.

There are many application areas for packet scheduling algorithms, ranging from detailed quality of service guarantees for individual application flows [2] to service assurances for aggregates [3]. In each of these scenarios, a more precise scheduler translates into more efficient resource usage in relation to the "quality" of the service guarantees. Because of the significant complexity and execution cost of packet schedulers, the sweet spot of network and capacity planning has been in configurations with very simple schedulers, so far. However, if truly low-cost and sophisticated packet schedulers would become available, they would likely be useful in many different ways.

The main criteria for evaluating the "scheduling quality" of a packet scheduler are the *delay bound*, especially in a form that can be used to determine an end-to-end delay bound as shown in several analytical frameworks [3], [4], [5], [6], as well as two fairness measures. *Relative fairness* (introduced in [7]) denotes the capability of a scheduler to distribute excess capacity between different sessions in proportion to their allocated service rates. *Worst-case fairness* (introduced in [8] and refined in [9]) expresses the maximum deviation from perfect GPS scheduling. While the delay bound only characterizes how far the actual service for a session can be *behind* the ideal GPS scheduler, worst-case fairness essentially provides an integrated bound on how far *ahead* or *behind* the actual service can be. Fairness thus also describes the burst characteristics of the service allocation in relation to the ideal smooth service of GPS. The key metric for describing the quality of these service characteristics as well as the computational complexity of a packet scheduler is the asymptotic relation between the respective characteristic and the number of flows in the system, which is described as constant, logarithmic, or linear. For example, constant delay describes the property that the delay bound is independent of the number of flows.

This paper presents a data structure called *Interleaved Stratified Timer Wheels* (ISTW) in combination with new algorithms for the *findNext* and *transferEligible* operations. This permits to construct a novel packet scheduler termed *Stratified/Interleaved Worst-case Fair Weighted Fair Queueing* (SI-WF^{2}Q). It has constant complexity, and constant fairness and delay characteristics in all relevant dimensions. The ISTW data structure is used as a compact and efficient priority queue that enables the virtual traffic shaping necessary for constant worst-case fairness. Constant complexity in this context is defined as amortized execution cost over a certain amount of input traffic. Amortized cost can be translated into buffering and as such, additional delay. In contrast to [10] however, the amortization period for SI-WF^{2}Q is limited and shown to be practical. In particular, the worst-case execution cost for most per-packet operations is proportional to the size of the corresponding packet. One processing step requires a longer amortization period, but the relevant actual duration of this period diminishes with increasing link speeds.

The rest of the paper is organized as follows. In the next section, existing work is examined and its relation to this work is discussed. In Section III, the ISTW data structure and access algorithms are introduced, which facilitate the scheduling algorithm. Section IV establishes the key characteristics of the resulting packet scheduler and provides a complexity analysis. To reaffirm and further illustrate the scheduling properties, some simulation results are presented in Section V. Section VI presents a summary of the properties and limitations of this scheduler and is followed by some conclusions and future work items in Section VII.

This paper is an extended version of [11].

II. Related Work

Existing GPS emulation algorithms can be classified as timestamp schedulers, round-robin schedulers, and hybrid versions. Different schedulers provide different combinations of the aforementioned scheduling quality characteristics, while none of the existing proposals is optimal in all quality dimensions and also of low complexity and execution overhead. There is a class of fundamentally different schedulers, termed *Service Curve Schedulers* [12], [13], which can provide delay bounds independently of throughput guarantees. These schedulers are inherently more complex than GPS emulation schedulers, so it seems hopeless to think about efficient implementation before solving the GPS emulation problem. Hence, although the techniques presented here may be applicable to such schedulers, this is out of scope for this paper.

A. Timestamp Schedulers

Timestamp schedulers approximate GPS behaviour by simulating the virtual system time in the equivalent GPS system. The respective start and/or finish times of packets in the reference GPS system are used to decide the order in which packets receive service. By the same token, the simulated virtual time is used as a starting point for newly arriving flows, which is necessary to achieve at least some bound on unfairness and burstiness. This challenge is well-known since the earliest proposals for proportional packet scheduling [14], [15]. Note that the order of packets within a flow is not affected by the scheduler operation. Therefore, only the first packet in each flow's queue needs to be considered for scheduling and the term "flow timestamp" is often used when referring to the packet timestamp at the head of the flow's queue.

B. Worst-case Fair Weighted Fair Queueing (WF^{2}Q)

An optimal packet approximation of GPS is presented in [9] and termed *Worst-case Fair Weighted Fair Queueing* (WF^{2}Q). The deviations from GPS scheduling are bound by strictly rate-dependent values for the respective flow (or two flows in the case of relative fairness) with provably optimal coefficients. In particular, all scheduling errors are independent of the number of flows in the system. Earlier attempts at approximating GPS, such as proposed in [1], [7], [14], [16], incur a potentially linear deviation from GPS scheduling, in terms of either fairness or delay behaviour. In fact, it turns out that when considering only packet start times for scheduling, the startup delay cannot be limited effectively and the delay bound depends on the number of flows in the system. When scheduling packets only by increasing finish times, packet bursts and unfairness can occur, also bound only by the number of flows.

Conceptually, the WF^{2}Q algorithm works by combining both criteria, start and finish time. In a first shaping step all eligible packets are selected, that is all packets with a start time not later than the current system virtual time. From all these packets, the one with the smallest finish time is sent next. This packet selection policy, termed *Start-eligible Earliest Finish-time First* (SEFF) ensures tight bounds for all quality indices. The authors of [17] independently arrive at the same conclusion that the combination of traffic shaping and finish-time service results in optimal scheduling characteristics. While [17] only describes a very simplified implementation in the context of fixed-size packet networks (ATM in this case), the original WF^{2}Q proposal in [9] is not at all concerned with algorithmic complexity or execution overhead.

C. WF^{2}Q Approximation

There are two proposals for implementing an approximation to WF^{2}Q with lower complexity: WF^{2}Q+ [18] and *Leap Forward Virtual Clock* (LFVC) [10]. The work in [17] contains a similar concept, but does not elaborate on all details. In general, all SEFF-based algorithms contain three parts that are relevant for their execution overhead and complexity:

- Flows are sorted according to timestamps.
- The SEFF policy requires consideration of both the start and the finish timestamp for the scheduling decision.
- The virtual time of the GPS reference system needs to be simulated or approximated.

Sorting and priority queues are among the best-studied problems in Computer Science. Without further restrictions, maintaining a sorted container has $O(\log N)$ complexity in the number of elements. In the context of GPS emulation schedulers, however, it can be a very acceptable trade-off to use rounded time values (and incur some additional scheduling error) in exchange for a finite universe of sorting values, which enables more efficient solutions to the sorting problem. For example, the van Emde Boas priority queue [19] has $O(\log \log N)$ access complexity for insertion, removal and finding the lowest value. Similarly, a timer wheel [20] could operate in $O(1)$ for insertion or removal and, in combination with hierarchical bitmaps and a priority encoder of width K, with $O(\log_K N)$ complexity for searching for lowest value (see Section 5 in [21] for a brief discussion). In both cases, a finite time horizon must be assumed, which translates into a maximum specifiable inter-arrival time of packets. Since the maximum packet inter-arrival time is part of the lower bound on the startup delay, one can assume that there is an upper limit to this value and it will not change in future networks. Then, if the desired delay precision is also fixed in terms of wall-clock time, one could argue that these algorithms operate with constant complexity in all relevant dimensions.

Unfortunately, the SEFF policy makes the above considerations somewhat irrelevant. It basically requires keeping flows in a two-dimensional container where they are sorted by both their start *and* finish times. This approach has been chosen for WF^{2}Q+, but inevitably requires a tree-based data structure and consequently $O(\log N)$ access complexity. Alternatively, flows can be kept in two one-dimensional containers, as proposed for LFVC, and exploit the lower access complexity discussed above. However, this two-container solution requires the transfer of all newly eligible flows from one container to the other in between the processing of two consecutive packets. Since all flows may end up with the same or very close start times, this number cannot be limited and effectively results in $O(N)$ worst-case complexity. While the transfer cost could be amortized over the number of packets transferred, this amortization would require $O(N)$ buffer space and result in a corresponding scheduling error.

D. Virtual Time

Traditionally, the precise simulation of GPS virtual time has been considered as being an operation with linear complexity, since it needs to keep track of all changes in the set of flows backlogged in the GPS reference system. A recent proposal [22] allows for the exact simulation of GPS virtual time with $O(\log N)$ algorithmic complexity in the number of flows. Independent analysis in [23] shows $O(\log N)$ to be a lower bound. Both WF^{2}Q+ and LFVC (along with other algorithms) use a simpler approximation of GPS virtual time. Basically, the approximated virtual time progresses with real time during actual service, that is, it is incremented by the duration of the current packet at each scheduling step. If however the smallest start time of all backlogged flows (which is readily available in WF^{2}Q+ and LFVC) is larger than the current virtual time, the virtual time jumps forward to this minimum start time.

This approximation of GPS virtual time has some negative effects on the scheduling quality of WF^{2}Q+ and LFVC, as pointed out in [22]. Under certain circumstances, the approximation of virtual time could lead to unfairness and burstiness being linear in the number of flows. This observation is not a contradiction of the findings for WF^{2}Q+ and LFVC, but results from different assumptions. Normally, GPS emulation schedulers are considered in a context where some kind of delay guarantees are sought. Such delay guarantees can only be given if the sum of rates of all flows does not exceed the link capacity. In terms of the GPS definition, this denotes a situation where the sum of weights is less than or equal to 1. In this case, all previous results about WF^{2}Q+ and LFVC hold, and burstiness is independent of the number of flows. If this restriction is removed, then the observations reported in [22] become an issue. However, it is somewhat questionable whether any application scenario requires the support for sum of weights exceeding 1. Certainly, all scenarios that aim at providing some form of delay guarantee do not qualify.

Along the same lines, there is another seeming contradiction between the results of [24] and the results presented for SI-WF^{2}Q. This discrepancy is also rooted in different assumptions about weights. In the most general case, the lower complexity bounds established in [24] presumably hold (this author has not personally checked them). However, in a somewhat restricted but very realistic scenario, the sum of weights is limited by 1, as discussed above. Furthermore, the maximum spread between the highest and lowest service rate is limited. For such a scenario, SI-WF^{2}Q is a better scheduling solution with a small and constant execution overhead.

E. Low Complexity Implementation

A number of techniques for the efficient implementation of timestamp schedulers are presented and discussed in [25]. For the particular case of fixed packet sizes in ATM networks, the article presents an implementation of WF^{2}Q+ with constant execution overhead. In the case of variable packet sizes, a different solution is presented, which technically can be regarded as having $O(1)$ complexity in the number of flows, but there are shortcomings. The scheduler implementation uses stratification in the virtual service time of packets to reduce the complexity of the one-container solution referred to in [18]. This results in a number of stratified groups which is logarithmic to the ratio of the maximum over the minimum supported service rate. For example, if the system were to support service rates between 16 Kbit/s and 40 Gbit/s at packet sizes ranging from 64 to 1500 bytes, this would result in 26 stratified groups. The algorithm then specifies that between each scheduling step, it is necessary to inspect the start and finish times of the front flow in each group to determine the next one to receive service under the SEFF policy. This sequence of comparisons is a nontrivial and costly operation and is hardly possible within the strict timing bounds of high-speed links. In other words, this sequence of comparisons introduces too high an absolute constant overhead per scheduling step. Further, the proof of the main Theorem 2 in [25] uses the condition that a flow always has an eligible packet during a time period, which restricts the general applicability of the result. The analysis of SI-WF^{2}Q arrives at basically the same conclusion and thereby remedies this situation.

F. Round Robin and Hybrid Schedulers

Round robin schedulers take a fundamentally different aim at emulating GPS scheduling. Instead of timestamp computations and sorting, service slots are assigned in some modified round robin fashion. This dramatically reduces the algorithmic complexity of such schedulers, but most early proposals suffer from rather large error terms in their fairness and delay properties. A more recent example, *Smoothed Round Robin* (SRR) [26], uses a fixed weight matrix to achieve very low computational complexity. Its relative fairness only depends on the order of the weight matrix and is thus independent of the number of flows. However, the weight matrix cannot be changed easily and the delay and worst-case fairness of SRR is linear in the number of flows. Recent proposals, such *Stratified Round Robin* (STRR) [27], *Fair Round Robin* (FRR) [28], and *Group Round Robin* (GRR) [29], use flow stratification along service rates and a two-level scheduling hierarchy to solve the problem of dynamically adding and removing flows.

The critical parameter determining the trade-off between quality and complexity of a round robin scheduler is the size of the quantum used for each scheduling round. Any round robin scheduler has an error term of $\frac{a \times \text{MTU}}{\text{flow rate}}$ due to the minimum quantum. Such a scheduling error poses a problem for low-rate flows with small packet sizes, such as voice. Further, if the quantum is chosen too small, this can lead to multiple processing steps without output (*slip processing*) and thus break $O(1)$ complexity. On the other hand, a larger quantum results in even larger error terms.

STRR uses a large quantum and has perfect $O(1)$ complexity. However, the algorithm's general delay bound and worst-case fairness is linear in the number of flows. FRR does not specify a particular quantum, but any quantum that can avoid linear scheduling errors inevitably leads to slip processing in the round robin loop. This effectively breaks $O(1)$ complexity. GRR is a general technique for hierarchical scheduling using round robin as intra-group scheduler. The quantum problem is approached differently then in other hierarchical round robin schedulers. GRR allocates a large quantum to each group, but interleaves group service according to the inter-group scheduler. While this results in the best scheduling properties of all round robin schedulers, it poses the risk of breaking $O(1)$ complexity through slip processing: A flow that becomes non-backlogged cannot be removed from the round robin list immediately. Instead, if it is still non-backlogged during the next round, it is considered departed and effectively removed from the list. However, the number of departed flows may be arbitrarily large in any round and therefore, may result in $O(N)$ processing steps between the processing of two consecutive packets.

G. Hierarchical Scheduling

Hierarchical scheduling allows for increased control over link sharing and resource allocation. Some of the shortcomings of the early fair-queueing schedulers become very apparent in the context of hierarchical packet scheduling [18]. Hierarchical rate-based scheduling invariably increases the delay bounds for leaf classes and as such, service curve schedulers such as SCED [12] and HFSC [13] are more suitable to achieve both link sharing and tight delay goals at the same time, albeit with increased complexity. For the purpose of this work, we do not argue in favour or against the usefulness of hierarchical scheduling. However, as noted from [18], hierarchical configurations can be regarded as an excellent litmus test for the fairness characteristics of a packet scheduling algorithm. It is strictly for this reason that the simulation experiments use hierarchical scheduling.

III. Packet Scheduler Description

The ISTW data structure can be used as a sorted timer container with desirable properties in the context of a timestamp packet scheduler. Essentially, the following proposal describes a modified version of LFVC scheduling [10] with the original shortcomings being addressed by the new data structure and access operations.

A. SI-WF^{2}Q System

As with other packet approximation algorithms of GPS, the system is defined as a set of flows $\mathcal{F}$. Each flow i is assigned a relative rate r_i, such that

$$\sum_{i \in \mathcal{F}} r_i \leq 1. \tag{1}$$

Further, the start time $S_i(\cdot)$ and finish time $F_i(\cdot)$ of a flow i are defined as follows, based on the system virtual time $V(\cdot)$:

$$S_i(t) = \begin{cases} \max(V(t), F_i(t-)) & \text{activation of flow i} \\ F_i(t-) & \text{next packet of flow i} \end{cases} \tag{2}$$

$$F_i(t) = S_i(t) + \frac{l_i}{r_i} \quad \text{for next packet with length } l_i \tag{3}$$

The service time of a packet $\frac{l_i}{r_i}$ is termed *virtual packet time*. The system uses rounded timestamps for scheduling (see next sections for details) and the system virtual time $V(\cdot)$ is based on the rounded start time function $\hat{S}_i(\cdot)$ (equivalent to the definitions in [10] and [25]) as

$$V(t+\tau) = \max(V(t)+\tau, \min_{i \in \mathcal{B}(t+\tau)} (\hat{S}_i(t+\tau))) \tag{4}$$

with $\mathcal{B}(t+\tau)$ being the set of all flows that are backlogged at time $t+\tau$. In simplified terms, among all flows i with $\hat{S}_i(t) \geq V(t)$, the flow with the smallest rounded finish time is served next. The virtual time is updated immediately after a packet is selected for service. We assume that the reference time t elapses at the speed of link transmissions and measure both reference and virtual time in bytes. We omit the reference time where it is not needed.

B. Problem Statement

LFVC proposes uniform rounding of timestamps and use of the van Emde Boas priority queue for sorting. Alternatively, one could imagine using a timer wheel enhanced by hierarchical bitmaps and a priority encoder. In both cases, the algorithmic complexity is quite low (and can technically be regarded $O(1)$, as discussed in Section II), but the absolute execution overhead is non-trivial. In [10], for example, 200-300 instructions are reported per packet operation. Furthermore, both the van Emde Boas data structure and hierarchical bitmaps consume a considerable amount of memory, which likely exceeds the amount of on-chip memory. In contrast, SI-WF^{2}Q uses stratified timestamp rounding. This reduces the corresponding execution overhead to a few instructions and keeps all relevant state in a compact data structure, suitable for a small memory footprint.

LFVC implements the SEFF packet selection policy by storing each flow in either of two containers, termed Low and $High$, depending on the eligibility of the respective front packet. If the flow's start time is later than the current virtual time, the flow is kept in Low, which is sorted by start times. Otherwise, the flow is stored in $High$, which is sorted by finish times. After each scheduling step, the eligibility of flows in Low is tested. All flows that have become eligible need to be transferred to $High$. This may result in $O(N)$ transfer operations, as illustrated in Figure 1. Note that LVFC uses an implicit notion of start times, which is results in slightly

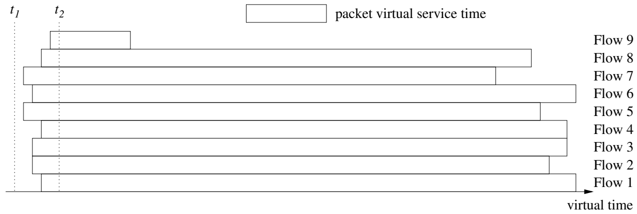

Fig. 1. Flow Transfer Problem in LFVC

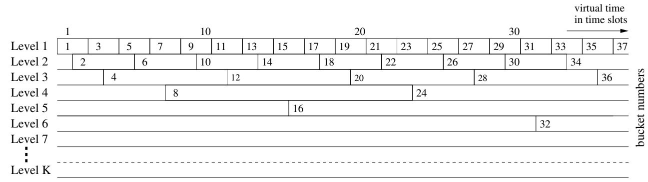

Fig. 2. Interleaved Stratified Timer Wheels (ISTW)

different scheduling characteristics, but does not change the big picture. Consider the case where the virtual time changes from t_1 to t_2 after a packet of corresponding size has been scheduled for transmission. At that point, all flows need to be examined to ensure that Flow 9 is transferred into $High$, which is necessary to serve it before its impending deadline. With SI-WF^{2}Q, it is possible to sensibly choose between different flows to control the number of transfers at each time while also ensuring that each flow is transferred "on time".

C. Data Structure

The ISTW data structure consists of a set of timer wheels, each representing a group of flows with similar weight allocations. Comparable to [27], [28], [29], but different from [25], flows are "stratified" into flow classes based on their relative rates. Formally, the set of flows $\mathcal{F}$ is partitioned into K flow classes G_k for level $k = 1, ..., K$ and defined as

$$G_k = \left\{ i : \frac{1}{2^{k-1}} \geq r_i > \frac{1}{2^k} \right\}. \tag{5}$$

Each of these classes is represented by a separate timer wheel, which consists of a fixed number of buckets. We denote the smallest possible packet size with λ and the maximum packet size with L. In the context of the ISTW data structure, virtual time is expressed in time slots, each of which covers the time equivalent to send λ bytes on the link. A timer wheel bucket at level k then covers 2^k time slots, which represents the virtual transmission time of a minimum size packet at the lower-end service rate of that level. According to (5), the maximum virtual packet time of any packet at level k is $2^k L$. The ISTW data structure must at least support this time horizon for each class. Therefore, the timer wheel at each level contains $\mathcal{L} \geq \lceil \frac{L}{\lambda} \rceil$ buckets. Buckets are assigned a unique number establishing a global ordering between buckets, as shown in Figure 2. The global bucket number $h_k(j)$ is equivalent to the first slot number covered by the bucket and is computed for any time slot j and level k as

$$h_k(j) = 2^k \left\lfloor \frac{j - 2^{k-1}}{2^k} \right\rfloor + 2^{k-1}. \tag{6}$$

Given a global bucket number h, the level k can be computed by finding the smallest bit set in h as

$$k = \mathrm{ffs}(h). \tag{7}$$

Only the next $\mathcal{L}$ buckets need to be present in each level at any time, so the timer wheels can wrap around. We denote the set of flows stored in bucket x with E_x and the number of elements in this set with $\parallel E_x \parallel$. Note that $\parallel E_{x_k} \parallel < 2^k$ at any time, since according to (5), there must be less than 2^k flows with a rate greater than $\frac{1}{2^k}$. Bucket 0 and all buckets h with $\mathrm{ffs}(h) > k$ do not belong to any level and are ignored during processing.

Each timer wheel bucket stores flows in an unsorted FIFO list. This corresponds to rounded timestamps, first by λ and second by the level-dependent time coverage of buckets. According to the findings in [10] and [25], rounded timestamps work well, if the start time is rounded down while the finish time is rounded up. In the case of SI-WF^{2}Q, this is accomplished by adjusting the start and finish time slots before computing the respective bucket numbers according to (6) as

$$\hat{s}_i = h_k(\frac{S_i}{\lambda} - 2^k) \tag{8}$$

and

$$\hat{f}_i = h_k(\frac{F_i}{\lambda} + 2^k). \tag{9}$$

for flow i in class k. The corresponding timestamps are

$$\hat{S}_i = \hat{s}_i \lambda, \quad \text{and} \quad \hat{F}_i = \hat{f}_i \lambda. \tag{10}$$

Given a current virtual time V_c, the current virtual time slot v_c is always computed as $v_c = \lfloor \frac{V_c}{\lambda} \rfloor$. Note that these formulae can be implemented with a few simple arithmetic operations, if λ is a power of 2.

To explain the above rounding, consider a flow in class k with a start time in the interval $[b, b + 2^k]$ where h is a global bucket number for level k. If the start time would simply be rounded down to h, the flow may end up at the end of the FIFO list and it may only be processed near time $h + 2^k$, which is too late (usage of Lemma 3/T2a in proof of Lemma 5). Therefore, the start time must be rounded down to the previous bucket covering $[b - 2^k, b]$ as in (8), so that the flow is surely processed before its actual start time. For finish times, the opposite consideration applies and results in (9).

The ISTW data structure has two key characteristics that make it very suitable for building an efficient packet scheduler. First, the stratification of timer wheels amounts to timestamp rounding with an error proportional to the service rate of flows. This results in a relatively small number of buckets, but limits the corresponding scheduling error. Second, the interleaving of stratified timer wheels facilitates browsing through flow classes, such that the frequency of visits is roughly proportional to the service rate. This is essential for a controlled transfer of flows from Low to $High$.

D. Scheduling Algorithm

The SI-WF^{2}Q scheduler is built similar to the LFVC scheduler. The ISTW data structure is used for both the Low (start

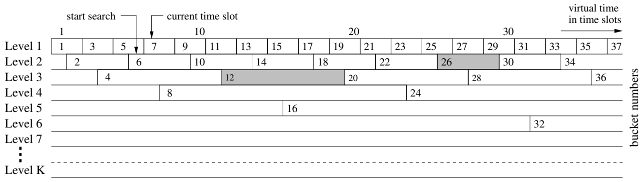

Fig. 3. Searching ISTW

time sorting) and the $High$ (finish time sorting) container. The *findNext* operation is used to search for the lowest timestamp in an ISTW container, while the *transferEligible* operation performs regulated but timely transfers from Low to $High$.

1) Lowest Timestamp Search - findNext: To facilitate an efficient search for the smallest element, a bitmap `anybits` of size K is kept, such that the bit at position k indicates whether any element is currently stored at level k. Searching starts by determining the lowest bit set in this bitmap. This denotes the *search level* k_c. Applying (6) with parameter k_c to the current virtual time slot v_c gives the starting bucket for the second step. Since there are no elements in any level smaller than k_c, the second step is a linear search with an increment of 2^{k_c-1}. As an example, consider Figure 3. Assume that the current virtual time slot is 7. The lowest set bit in `anybits` indicates 2 as the smallest level with an element. Assume this element is stored in Bucket 26. The linear search then starts searching at Bucket 6, which is the bucket in Level 2 covering Slot 7. The linear search browses through buckets by incrementing in steps of 2. If an "earlier" bucket at a greater level contains an element, this is found appropriately. In this example, if Level 3 contains an element in Bucket 12, then this is found appropriately - before the search reaches Bucket 26. The following pseudo-code describes the operation:

```
function findNext( inout curr_slot ) {
  k = ffs(anybits);
  curr_slot = h_k(curr_slot);
  while (getBucket(curr_slot,k).empty()) {
    curr_slot += 2^(k-1);
    k = ffs(curr_slot);
  }
  return getBucket(curr_slot,k).front();
}
```

2) Regulated Flow Transfer - transferEligible: By applying (6) to the current virtual time slot, it is possible to identify the "current" bucket $h_k(v_c)$ for level k. A bitmap `frontbits` of size K is maintained to indicate which levels have elements stored in their respective current buckets. When the virtual time is incremented by the size of last packet, the virtual time slot v_c is incremented accordingly. For each increment by one slot, the following operations are executed:

- If the bucket with number v_c is not empty, compute the level number as ffs(v_c) and set its bit in `frontbits`.
- Find the lowest bit k in `frontbits`. If found, transfer one flow from the current bucket of level k to $High$.

Essentially, if several flows are eligible for transfer, the second step ensures that flows with a higher service rate are transferred first. However, as shown below in Lemma 1, no flow is left behind. Instead, the transfer operation exploits the fact that flows at different rate levels have different timing requirements. The operation is illustrated by pseudo-code:

```
function transferEligible() {
  curr_slot = virt_time / λ;
  virt_time += last_packet_size;
  while (curr_slot ≤ virt_time / λ) {
    k = ffs(curr_slot);
    if (!getBucket(curr_slot,k).empty())
      set_bit(frontbits,k);
    if (frontbits > 0) {
      k = ffs(frontbits);
      f = getBucket(curr_slot,k).front();
      transferToHigh(f);
      if (getBucket(curr_slot,k).empty())
        clear_bit(frontbits,k);
    }
    curr_slot += 1;
  }
}
```

3) Packet Service - chooseFlow: During each processing step, the next flow is selected from the $High$ container using *findNext*. While this packet is transmitted, the scheduler transfers eligible packets from Low to $High$ using *transferEligible*. If $High$ is empty, the next start time is searched in Low using *findNext*. The virtual system time is updated according to (4). The corresponding pseudo-code is given below:

```
function chooseFlow() {
  old_slot = virt_time / λ;
  Low.transferEligible();
  if (High.empty()) {
    curr_slot = virt_time / λ;
    f = Low.findNext( curr_slot );
    transferToHigh(f);
    virt_time = curr_slot * λ;
  }
  return High.findNext( old_slot );
}
```

IV. Analysis

The basic scheduling properties of SI-WF^{2}Q are analyzed using a similar proving strategy and adapted proofs from [10]. In a second step, the algorithmic complexity and its relation to scheduling properties is studied.

A. Scheduling Properties

In contrast to WF^{2}Q+ and LFVC, SI-WF^{2}Q does not strictly adhere to the SEFF policy, but instead mediates transfers from Low to $High$ with the *transferEligible* operation. Lemmas 1 and 2 establish that each flow is transferred "on time", which is a crucial prerequisite to use the proving strategy from [10].

Lemma 1: In Low, for each level k, all buckets earlier than the current bucket are empty. Formally:

$$\parallel E_{x_k} \parallel = 0 \text{ for all } k, x_k \text{ with } x_k < h_k(v_c) \tag{11}$$

Proof: A bucket h_k at level k covers a virtual time period equivalent to 2^k time slots. In other words, each level is "visited" once every 2^k slots by *transferEligible* and each bucket h_k remains as the current bucket (and in `frontbits` if necessary) for 2^k time slots. We are only concerned with time periods during which `frontbits` $\neq 0$, because otherwise, all current buckets are empty and the Lemma holds. For each time slot during such a busy period, exactly one flow is transferred from Low to $High$ by *transferEligible*. Each flow can only occur once in each bucket of Low. While two consecutive start timestamps might fall in the same bucket, this also results in identical rounded start times and consequently, the flows simply stays in $High$ for the second packet. We prove by induction over k that for an arbitrary time period of 2^k slots, at most 2^k timestamps of flows in levels $1..k$ are ever encountered. Flows at levels $k+1..K$ can be ignored when considering level k, since they have lower priority in `frontbits`. Because of (1), this shows that each bucket is drained, before switching to the next bucket in that level. Define the union of flow classes up to level k as U_k with

$$U_k = \bigcup_{i=1}^{k} G_k. \tag{12}$$

Induction Hypothesis: For an arbitrary $v \in \mathcal{N}_0$,

$$\sum_{x=v}^{v+2^k-1} \parallel E_x \parallel \leq \ 2^k \sum_{i \in U_k} r_i + \parallel E_z \parallel \ \text{ with } z > k. \tag{13}$$

Base Case ($k = 1$): One of the buckets E_v and E_{v+1} belongs to level 1. Assume without loss of generality E_v. Then E_{v+1} belongs level z with $z > 1$. If E_v contains an element, U_1 is not empty and the inequality evaluates to

$$1 + \parallel E_{v+1} \parallel \leq \ 2r_i + \parallel E_{v+1} \parallel \tag{14}$$

and otherwise (no element in E_v) to

$$\parallel E_{v+1} \parallel \leq \parallel E_{v+1} \parallel . \tag{15}$$

Because $r_i > 0.5$ for $k = 1$, (14) is correct, while (15) is trivial. Therefore, the hypothesis is verified for $k = 1$.

Inductive Step: Assume the hypothesis holds for k. Then

$$\sum_{x=v}^{v+2^{k+1}-1} \parallel E_x \parallel = \sum_{x=v}^{v+2^k-1} \parallel E_x \parallel + \sum_{x=v+2^k}^{v+2^{k+1}-1} \parallel E_x \parallel \tag{16}$$

$$\leq 2^k \sum_{i \in U_k} r_i + \parallel E_{z_1} \parallel \ + 2^k \sum_{i \in U_k} r_i + \parallel E_{z_2} \parallel \tag{17}$$

$$= 2^{k+1} \sum_{i \in U_k} r_i + \parallel E_{z_1} \parallel + \parallel E_{z_2} \parallel \ \text{ with } z_1, z_2 > k \tag{18}$$

$$\leq 2^{k+1} \sum_{i \in U_{k+1}} r_i + \parallel E_{z_2} \parallel \ \text{ with } z_2 > k+1. \tag{19}$$

The first equation simply splits the 2^{k+1} time slots into two terms. In the transformation step to (17), the induction hypothesis is applied and the transformation to (18) rearranges the terms. The last step uses the following consideration: For any 2^k consecutive slot numbers, 2^n buckets belong to level $k-n$ for n in $1..k$, while one bucket belongs in to level $x > k$. This can easily be verified using (6) and Figure 2. Consequently, one of the buckets E_{z_1} and E_{z_2} belongs to level $k+1$. Assume without loss of generality E_{z_1}. From (5) follows

$$\parallel E_{h_{k+1}} \parallel < 2^{k+1} \sum_{i \in G_{k+1}} r_i. \tag{20}$$

Since $U_k \cup G_{k+1} = U_{k+1}$, inserting (20) into (18) leads to (19). Thus, the induction hypothesis is verified for $k+1$. ■

The next lemma proves that no flow is transferred to $High$ too late. Essentially, each flow is transferred within the time frame of one bucket in its level, which in turn is equivalent to the virtual transmission time of a minimum size packet at the lower-end service rate of that level. In fact, each flow is transferred before its original (non-rounded) start time. Define the *effective start time* as the virtual time at which a flow is transferred from Low to $High$ and denote it with e.

Lemma 2: The effective start time e_i of flow i at level k is bound by

$$\hat{S}_i \leq e_i \leq (\hat{S}_i + 2^k\lambda) \leq S_i \leq (\hat{S}_i + 2^{k+1}\lambda). \tag{21}$$

Proof: The relation between $\hat{S}_i$ and S_i follows directly from (8) and (10).

Lower Bound of e_i: According to (8), a flow with start time S_i is stored in bucket $\hat{s}_i$. During *transferEligible*, this bucket is inspected not earlier than at virtual time $\hat{S}_i$ and only afterwards can the flow be transferred. If bucket $\hat{s}_i$ is reached by start-time searching (cf. Section III-D), the virtual time is also set to $\hat{S}_i$.

Upper Bound of e_i: Lemma 1 proves that each bucket is emptied, before the next bucket in the same level is inspected by *transferEligible*. The next bucket has the number $\hat{s}_i + 2^k$ and is inspected at $\hat{S}_i + 2^k\lambda$. ■

Lemma 3: Let L_i denote the maximum packet size for a flow i. The following timestamp invariants hold.

T1a: If flow i is in $High$, then $S_i \leq V + 2^{k+1}\lambda$

T1b: If flow i is in $High$, then $F_i \leq V + \frac{l_i}{r_i} + 2^{k+1}\lambda$

T2a: If flow i is in Low, then $S_i \geq V$

T2b: If flow i is in Low, then $F_i \geq V + \frac{l_i}{r_i}$

T3a: For head packet size l_i: $S_i \leq V + \frac{L_i}{r_i} + 2^{k+1}\lambda$

T3b: For head packet size l_i: $F_i \leq V + \frac{l_i}{r_i} + \frac{L_i}{r_i} + 2^{k+1}\lambda$

Proof: T1 and T2 follow from Lemma 2 and the definition of rounded timestamps in (8) and (10).

T3, Case 1: Packet becomes new head of queue. T1 holds before service. Service of previous packet increases S_i and F_i by at most $\frac{L_i}{r_i}$.

T3, Case 2: Packet arrives at empty queue. If last $F_i \leq V$, then new $S_i = V$ and T3 follows. Otherwise same as Case 1: T1 holds before last service and increment by at most $\frac{L_i}{r_i}$. ■

The remaining lemmas and theorems are basically an adaptation of the corresponding analysis in [10].

Lemma 4: If, for all $V' \geq V$, every backlogged flow i with $\hat{F}_i \leq V'$ satisfies $l_i \leq (F_i - V)r_i$, i.e. $S_i \geq V$, then the following inequality holds:

$$\sum_{i:\hat{F}_i \leq V'} l_i + \sum_{i:\hat{F}_i \leq V'} (V' - F_i) r_i \leq V' - V \tag{22}$$

Proof: Since $l_i \leq (F_i - V)r_i$ for all flows, the set of flows satisfies

$$\sum_{i:\hat{F}_i \leq V'} l_i \leq \sum_{i:\hat{F}_i \leq V'} (F_i - V)r_i \tag{23}$$

$$= \sum_{i:\hat{F}_i \leq V'} (V' - V)r_i - \sum_{i:\hat{F}_i \leq V'} (V' - F_i)r_i \tag{24}$$

$$\leq (V' - V) - \sum_{i:\hat{F}_i \leq V'} (V' - F_i)r_i \tag{25}$$

The first step uses $V' - V - (V' - F_i) = F_i - V$ and the second step uses $\sum_{i:\hat{F}_i \leq V'} r_i \leq 1$, based on (1). ■

Lemma 5: At any virtual time V, the *Backlog Inequality* holds for all future virtual time values V', i.e.

$$\sum_{i:\hat{F}_i \leq V'} l_i + \sum_{i:\hat{F}_i \leq V'} (V' - F_i)r_i \leq L + V' - V \tag{26}$$

with l_i being the size of the first packet in flow i's queue. If a flow j is not backlogged, $l_j = 0$ and $F_j \geq V$, because of (2).

Proof: By induction over possible events in the system. The base case is trivial.

Packet Enqueue: Packet p arrives at the head of the queue. Denote the size of new packet with l_p, and flow rate and rounded finish time with r_p and $\hat{F}_p$. If $\hat{F}_p > V'$, nothing changes and the lemma holds. Otherwise, the first term $\sum_{i:\hat{F}_i \leq V'} l_i$ is incremented by l_p, but since the flow's finish time is incremented by the virtual packet time $\frac{l_p}{r_p}$, the second term $\sum_{i:\hat{F}_i \leq V'} (V' - F_i)r_i$ is decremented by l_p.

Packet Transfer: No relevant variables are changed.

Virtual Time Jump: After a virtual time jump $S_i \geq V$ is true for all flows in the system. Therefore, Lemma 4 holds.

Packet Service: Denote the size of the current packet with l_p, and flow rate and rounded finish time with r_p and $\hat{F}_p$. We have to distinguish two cases, depending on V' and $\hat{F}_p$.

Case 1: If $V' \geq \hat{F}_p$, then the first term $\sum_{i:\hat{F}_i \leq V'} l_i$ is decremented by l_p. Since V is incremented by l_p, the right-hand side of the inequality is decremented by the same amount. Therefore, the lemma holds. For the next packet in the queue, the case 'Packet Enqueue' applies.

Case 2: If $V' < \hat{F}_p$, then all flows with $\hat{F}_i \leq V'$ have $\hat{F}_i < \hat{F}_p$. All these flows must have been in Low, otherwise the current packet would not have been chosen. By Lemma 3/T2a, $S_i \geq V$ holds for all backlogged flows before the packet service and consequently, Lemma 4 holds then. Since packet service increments V (and decrements $V' - V$) by at most L, the lemma holds afterwards. ■

The next lemma establishes that all packets are served when the virtual time reaches their respective rounded finish time with an error term of one maximum packet size L.

Lemma 6: For any backlogged flow i in class k

$$F_i + 2^k\lambda \geq \hat{F}_i \geq V - L. \tag{27}$$

Proof: The proof of the right invariant is by contradiction. The only event that could lead to a violation of the lemma is serving a packet during a busy period. Assume that at V_1 the lemma holds. A packet p with rounded finish time $\hat{F}_1$ and length l is served and afterwards at V_2, there is a packet q with rounded finish time $\hat{F}_2$, such that $\hat{F}_2 + L < V_2$. Denote with S_1 and S_2 the corresponding start times. We need to distinguish three cases.

Case 1: Packet q is eligible at V_1. Then, $\hat{F}_2 \geq \hat{F}_1$ (both packets were eligible at V_1). Applying Lemma 5 with $V = V_1$ and $V' = \hat{F}_2$ results in:

$$l_p \leq \sum_{i:\hat{F}_i \leq \hat{F}_2} l_i \leq L + \hat{F}_2 - V_1 \tag{28}$$

$$V_2 - V_1 \leq L + \hat{F}_2 - V_1 \text{ since } l_p = V_2 - V_1 \tag{29}$$

$$V_2 \leq L + \hat{F}_2 \tag{30}$$

Case 2: A virtual time jump happens before serving q:

$$V_2 \leq S_2 \leq \hat{F}_2 \tag{31}$$

Case 3: Packet q becomes eligible between V_1 and V_2. Virtual time progresses by at most L, therefore:

$$\hat{F}_2 \geq S_2 \geq V_1 \geq V_2 - L \tag{32}$$

$$\hat{F}_2 + L \geq V_2 \tag{33}$$

The left invariant in the lemma follows from the definition of rounding in (9) and (10). ■

The last lemma establishes a bound between the virtual time V and real time R, expressed as bytes, assuming a finite output buffer and a fixed link capacity. This is a generalization of Lemma 4.5 in [10].

Lemma 7: Let I be the last time when the system was idle and the output buffer empty. Let J be the amount of virtual time "jumping" that has be done during the current busy period. Let B be the maximum output buffer. Then

$$R \leq V + I - J \leq R + B. \tag{34}$$

Proof: Assume that b be the amount of data from this busy period that is currently stored in the output buffer. Since the output buffer is drained in sequential order, independent of the scheduler, we can model it as fluid. We first prove

$$V + I - J = R + b. \tag{35}$$

The following events influence the variables in the invariant (35). When a packet is chosen for service and stored in the output buffer, V and b increase by the same amount. When the virtual time jumps, V and J increase by the same amount. During link transmission, R increases at the same speed at which b decreases. The fact that b is bound by B leads to the lemma. ■

Essentially, the above lemmas provide a lower and upper bound on service in relation to virtual time, as well as a bound between virtual time and real time. Further note that virtual time is never slower than real time. In particular, each packet p is served no later than $\hat{F}_p + L + B + I - J$. Traditionally, the output buffer has been considered to be of size L and empty when a busy period starts. Using these results, it is possible to derive bounds for the service characteristics of SI-WF^{2}Q. However, note that the bounds below are not necessarily tight. In particular, the error term L from Lemmas 5 and 6 is probably already included in the error term B from Lemma 7.

Theorem 1: (End-to-End Delay) SI-WF^{2}Q is a *Guaranteed Rate* (GR) scheduler as defined in [4], with error term β for a flow in rate level k with

$$\beta \leq L + 2^k\lambda + B \tag{36}$$

Proof: Let p_i^j and l_i^j denote the jth packet of flow i and its size. Let $A_R(p_i^j)$ denote the real-time arrival time of packet p_i^j. The guaranteed rate clock values are defined as is [4]:

$$GRC(p_i^j) = \max(A_R(p_i^j), GRC(p_i^{j-1})) + \frac{l_i^j}{r_i} \tag{37}$$

with $GRC(p_i^0) = 0$.

Denote with F_i^p the finish time of the jth packet. Let $I(p_i^j)$ and $J(p_i^j)$, respectively, be the values of I and J from Lemma 7 when service of p_i^j is completed. Let $I'(p_i^j)$ and $J'(p_i^j)$, respectively, be the values of I and J when p_i^j arrives at the head of its queue. We first prove that during a busy period

$$F_i^j + I(p_i^j) - J(p_i^j) \leq GRC(p_i^j) + B \tag{38}$$

We prove (38) by induction on j. The base case is trivial. The virtual time of a packet arrival is denoted by $A(p_i^j)$.

Inductive Step: Assume (38) holds for $j-1$. We need to distinguish two cases.

Case 1: $A(p_i^j) > F_i^{j-1}$. Then

$$F_i^j = A(p_i^j) + \frac{l_i^j}{r_i}. \tag{39}$$

Using Lemma 7 we can characterize the packet arrival time as

$$A(p_i^j) + I'(p_i^j) - J'(p_i^j) \leq A_R(p_i^j) + B. \tag{40}$$

Adding $\frac{l_i^j}{r_i}$ results in

$$A(p_i^j) + \frac{l_i^j}{r_i} + I'(p_i^j) - J'(p_i^j) \leq A_R(p_i^j) + B + \frac{l_i^j}{r_i}. \tag{41}$$

Rearranging the terms, using (39), and replacing A_R by a maximum, results in

$$F_i^j + I'(p_i^j) - J'(p_i^j) \leq \max(A_R(p_i^j), GRC(p_i^{j-1})) + \frac{l_i^j}{r_i} + B. \tag{42}$$

The right hand side shows the definition of GRC. Since we are in a busy period, $I'(p_i^j) = I(p_i^j)$ and $J'(p_i^j) \leq J(p_i^j)$. Therefore

$$F_i^j + I(p_i^j) - J(p_i^j) \leq GRC(p_i^j) + B \tag{43}$$

which establishes (38) for Case 1.

Case 2: $A(p_i^j) \leq F_i^{j-1}$. In this case, $F_i^j = F_i^{j-1} + \frac{l_i^j}{r_i}$. By induction hypothesis, we have

$$F_i^{j-1} + I(p_i^{j-1}) - J(p_i^{j-1}) \leq GRC(p_i^{j-1}) + B \tag{44}$$

Adding $\frac{l_i^j}{r_i}$ and replacing GRC by a maximum results in

$$F_i^j + I(p_i^{j-1}) - J(p_i^{j-1}) \leq \max(A_R(p_i^j), GRC(p_i^{j-1})) + \frac{l_i^j}{r_i} + B. \tag{45}$$

The right hand side shows the definition of GRC. Since we are in a busy period, $I(p_i^{j-1}) = I(p_i^j)$ and $J(p_i^{j-1}) \leq J(p_i^j)$. Therefore

$$F_i^j + I(p_i^j) - J(p_i^j) \leq GRC(p_i^j) + B \tag{46}$$

which establishes (38) for Case 2.

Using (38), we can now prove the lemma. A packet p_i^j is served no later than $F_i^j + L + 2^k\lambda$ (Lemma 6). At the end of transmission, the real time equals $F_i^j + L + 2^k\lambda + I(p_i^j) - J(p_i^j)$ by Lemma 7. By (38), this is bound by $GRC(p_i^j) + B + L + 2^k\lambda$. ■

Theorem 2: (Relative Fairness) For SI-WF^{2}Q, the relative fairness, as defined in [7], of any two flows i and j is bound by ζ with

$$\zeta_{i,j} \leq 2L + 3\max(\frac{L_i}{r_i}, \frac{L_j}{r_j}) + \min(\frac{L_i}{r_i}, \frac{L_j}{r_j}) + 12\lambda. \tag{47}$$

Proof: The earliest and latest start and finish times of a packet from flow i in level k that receives service after virtual time V follow from Lemma 6 and Lemma 3/T3:

$$S^{\min} \geq V - L - 2^k\lambda - \frac{L_i}{r_i}, \tag{48}$$

$$S^{\max} \leq V + \frac{L_i}{r_i} + 2^{k+1}\lambda, \tag{49}$$

$$F^{\min} \geq V - L - 2^k\lambda, \tag{50}$$

$$F^{\max} \leq V + 2 \times \frac{L_i}{r_i} + 2^{k+1}\lambda. \tag{51}$$

For an interval $[V_1, V_2]$ during which the flow is backlogged, this results in the following service bounds:

$$(F^{\min} - S^{\max})r_i \leq \text{service}_i \leq (F^{\max} - S^{\min})r_i. \tag{52}$$

For relative fairness, it is not necessary to convert the virtual times V_1 and V_2 to real time, since both flows are backlogged during the whole interval. The normalized service is then obtained by dividing the amount of service through the service rate and the maximum normalized difference between any two flows i and j is computed as

$$\zeta_{i,j} \leq (F^{\max} - S^{\min}) - (F^{\min} - S^{\max}) \tag{53}$$

The lemma follows from inserting (48) - (51) into (53) and using *max* and *min* appropriately. ■

Theorem 3: (Worst-case Fairness) Let $Q_i(p)$ be the new backlog of an arbitrary flow i at the time when packet p arrives. The time δ to clear this backlog is bound as

$$\delta \leq \frac{Q_i(p)}{r_i} + L + \frac{L_i}{r_i} + 3 \times 2^k\lambda + B. \tag{54}$$

Proof: Consider a packet p for flow i. Suppose that when p arrives, the packet at the head of flow i's queue is p_i^j. Suppose further that there are $m \geq 0$ packets in the queue in front of p, meaning $p = p_i^{j+m}$. Let R_1 denote the real arrival time and R_2 the real time when p's service is complete. Let V_1, V_2 (resp. F_1, F_2) denote the virtual time (resp. finish times) corresponding to the real times R_1 and R_2. Lemma 3/T3b guarantees the following inequality:

$$F_1 \leq V_1 + \frac{l_i^j}{r_i} + \frac{L_i}{r_i} + 2^{k+1}\lambda \tag{55}$$

or

$$-V_1 \leq -F_1 + \frac{l_i^j}{r_i} + \frac{L_i}{r_i} + 2^{k+1}\lambda. \tag{56}$$

The *Service Time* lemma gives the bound $V_2 \leq F_2 + L + 2^k\lambda$. Subtracting V_1 from V_2, i.e. adding (56) to the bound for V_2, results in

$$V_2 - V_1 \leq F_2 - F_1 + \frac{l_i^j}{r_i} + L + \frac{L_i}{r_i} + 3 \times 2^k\lambda. \tag{57}$$

Since flow i is backlogged during the interval $[F_1, F_2]$, we get $F_2 = F_1 + \sum_{n=1}^{m} \frac{l_i^{j+n}}{r_i}$, which can be inserted into (57):

$$V_2 - V_1 \leq \sum_{n=1}^{m} \frac{l_i^{j+n}}{r_i} + \frac{l_i^j}{r_i} + L + \frac{L_i}{r_i} + 3 \times 2^k\lambda. \tag{58}$$

The queue size $Q_i(p)$ satisfies

$$\sum_{n=1}^{m} \frac{l_i^{j+n}}{r_i} \leq Q_i(p). \tag{59}$$

Plugging this into (58) yields

$$V_2 - V_1 \leq \frac{Q_i(p)}{r_i} + L + \frac{L_i}{r_i} + 3 \times 2^k\lambda. \tag{60}$$

Using the bounds between virtual time and real time from Lemma 7, we can obtain a lower bound for R_1 and an upper bound for R_2:

$$V_1 + I - J \leq R_1 + B \text{ and } V_2 + I - J \geq R_2. \tag{61}$$

Therefore

$$\begin{aligned} R_2 - R_1 &\leq V_2 + I - J - (V_1 + I - J - B) & (62)\\ &= V_2 - V_1 + B & (63)\\ &\leq \frac{Q_i(p)}{r_i} + L + \frac{L_i}{r_i} + 3 \times 2^k\lambda + B. & (64) \end{aligned}$$

This concludes the proof. ■

The above theorems do not explicitly specify the link speed, because real time is modelled as the amount of data sent over the link and service rates r_i are expressed as relative to the link speed. To transform these formulae into the time domain, the bounds need to be divided by the link speed.

B. Algorithmic Complexity

We devise a logical system model as shown in Figure 4. Low and $High$ are the main flow containers, as described before. After a flow is selected from $High$, its front packet is appended to a link output queue, the flow is potentially re-inserted into either Low or $High$, the virtual time is updated, and *transferEligible* is executed. A separate *Link Output* thread pulls packets from the output queue and transmits them onto the link. Note that this system model does not necessarily require explicit thread programming, but an implementation can easily utilize multiple threads. The size of the link output queue is related to the error terms of the scheduler, since it determines the maximum deviation between virtual time and real time (see Lemma 7). As discussed below, it is necessary to add some amortization buffer to balance out fluctuations in the processing cost for individual packets. The operations that need to be considered for algorithmic complexity are *transferEligible* and *findNext*.

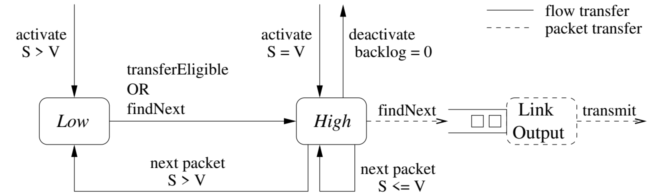

Fig. 4. Execution Model

1) Complexity of transferEligible: As described in Section III-D.3, *transferEligible* is invoked once for each packet being sent. The description in Section III-D.2 shows how it operates in proportion to the length of the packet in service. In particular, the number of processing steps is at most $\lceil \frac{2l}{\lambda} \rceil$ for a packet with length l and $\frac{2l}{\lambda}$ on average. Therefore, in relation to the number of flows and all other relevant dimensions, *transferEligible* executes with constant complexity.

2) Complexity of findNext for $High$: The *findNext* operation on the $High$ container amounts to a linear search from the current virtual time slot to the finish time slot of the eventually chosen packet in steps of 2^{k-1} slots, where k is the lowest level that has an element in the $High$ container. Since all flows in the $High$ container are eligible, the search starts from a slot equal or greater than the start time of the packet. Since start time slots are rounded down while finish time slots are rounded up, the number of traversed slots is limited by $\lceil \frac{2l}{\lambda} \rceil + 2$, and again $\frac{2l}{\lambda}$ on average, for a packet with length l. Because the *findNext* operation is executed *before* the packet is transmitted, an amortization buffer of the maximum packet size L is necessary to ensure continuous operation at an amortized complexity of $O(1)$.

3) Complexity of findNext for Low: The *findNext* operation on the Low container poses a somewhat more complicated scenario. We present a worst-case analysis here, which still results in a practical configuration. However, a better solution might exist, either by means of a tighter analysis or a different search mechanism. The maximum search distance for the next start time in Low is related to the size of the *previous* packet in the same class, while in the case of $High$ it is the *next* packet. Therefore, while the search cost in $High$ can be amortized immediately, this is not necessarily the case for searching Low. Assume that a large packet is followed by a smaller packet in the same class, but after serving the larger packet, the search is no fully carried out, because another eligible packet is found. Eventually the system might need to search this particular level, so a temporary output buffer of the size of the original packet is required to keep the link busy during this later search. Without further detailed analysis, this can happen at most once per stratified class and results in a worst-case amortization buffer of one a maximum packet size *per rate level*. The effect of such a buffer is expressed as error term B in the theorems presented in the previous section.

A simple analysis shows the limited real-world impact of the amortization buffer. The amount only grows logarithmically with the link speed, because the number of stratified classes only grows logarithmically. This results in a diminished abso-

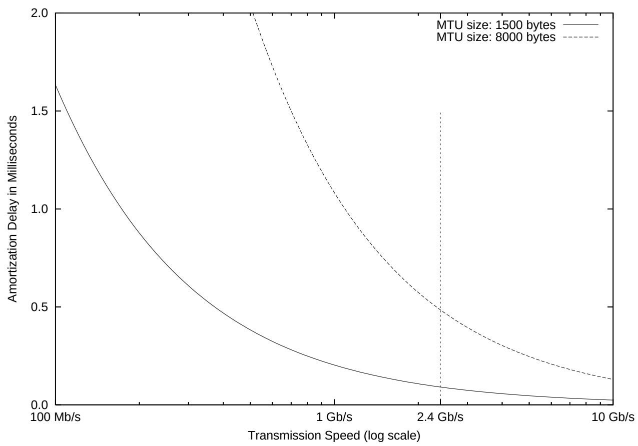

Fig. 5. Amortization Delay for *findNext* on Low

lute error, which can be quantified as

$$\text{delay} \leq \frac{\text{MTU} \times \log_2(\frac{\text{link speed}}{\text{minimum rate}})}{\text{link speed}}. \quad (65)$$

As an example, assuming a minimum rate of 8 Kbit/s, the resulting amortization delay for different MTU sizes is shown in Figure 5. While at a link speed of 100 Mbit/s, the amortization delay is on the order of several milliseconds, it is reduced below one millisecond very quickly. For a 1 Gbit/s link with an MTU size of 1500 bytes, the amortization delay already shrinks to approximately 0.2 ms. With an MTU size of 8000 bytes, the worst-case amortization delay is less than 0.5 ms for a link speed of 2.4 Gbit/s. Clearly, for even higher link rates in future networks, the amortization delay will be negligible.

V. Simulations

We have performed numerous simulations using ns-2 to verify the scheduling characteristics of the SI-WF^{2}Q scheduler. In all cases, the simulations confirm the predictions from Section IV-A. For this paper, the focus is on the worst-case fairness property of the new scheduler compared to WF^{2}Q+. We therefore reproduce the corresponding simulation experiments from Section VI.A in [18]. As mentioned before, hierarchical scheduling is an excellent litmus test to verify and illustrate the worst-case fairness properties of a packet scheduler. In contrast to the experiments in [18], we only compare to the non-shaped *Starting Potential Fair Queueing* (SPFQ) scheduler from [21]. Its worst-case fairness properties are comparable to the WFQ, SCFQ, and SFQ schedulers chosen in [18]. The experiment setup uses a simple dumbbell topology with a single bottleneck and multiple sender and receiver nodes. The bottleneck is configured with a hierarchy of scheduling classes as shown in Figure 6 and reproduces the setup from [18]. For each leaf class, there is a corresponding traffic source sending on average with the given rate. The PS-n sources send Poisson traffic with an on-period of 75 ms and an off-period of 25 ms. The BE-1 source is always backlogged and the RT-1 source sends CBR traffic with an on-period of 75 ms and an off-period of 25 ms. In one scenario, the CS-n sources do not produce any traffic and the background traffic

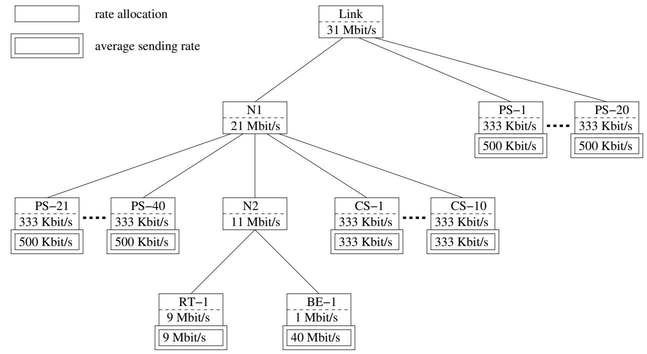

Fig. 6. Experiment Scheduling Hierarchy

is mostly uncorrelated. In a second scenario, the CS-n sources send CBR traffic and produce correlated background traffic. All packet sizes are set to 8000 bytes. The main observation parameter for these experiments is the devolution of the delay for the RT-1 traffic over the duration of the experiment.

The results of the experiments are shown in Figure 7. The graphs show the worst-case delay of the RT-1 traffic over intervals of 50 ms. For illustrative purposes, the SI-WF^{2}Q delay is shown excluding the amortization buffer discussed in Section IV-B. Similar to the original results in [18], a non-SEFF scheduler causes significant delay variations for the real-time traffic class. Correlated cross traffic further amplifies this characteristic. The superior worst-case fairness of WF^{2}Q+ results in significantly reduced delay fluctuations, independent of the type of cross traffic. As predicted by the analytical results, SI-WF^{2}Q provides basically the same service as WF^{2}Q+.

It is predicted in Section IV-A that the scheduling characteristics of SI-WF^{2}Q depend on the minimum packet size λ, which is used for timestamp rounding and establishing $O(1)$ execution complexity. For example, a realistic setting of λ could be 64 bytes and this number is used for the above simulation experiments. To illustrate this dependency, the SI-WF^{2}Q experiments are repeated with λ changed to 16384 and 32768 bytes. These values are chosen to exceed the packet size of 8000 bytes used in the experiments. The results are shown in Figure 8. It can be observed that the increase of λ results in an increased delay and delay variation, as predicted.

VI. Summary

SI-WF^{2}Q offers superior properties compared to all other known packet schedulers. The major benefits of SI-WF^{2}Q and other contributions in this paper are as follows:

- SI-WF^{2}Q has constant small scheduling errors, only depending on flow characteristics and constants depending on the link speed. In particular, there is no error term component of L_i/r_i, as in round robin schedulers.
- SI-WF^{2}Q can be executed in amortized $O(1)$ complexity with reasonable amortization bounds. This is the key improvement over the high speed proposal for WF^{2}Q+, which requires up to $2N$ comparisons (for N classes) to carry out the complete SEFF decision for each packet.
- The absolute execution overhead and memory footprint is small. At the core of the algorithm, the *findNext* operation only accesses the ISTW bitmaps frequently.

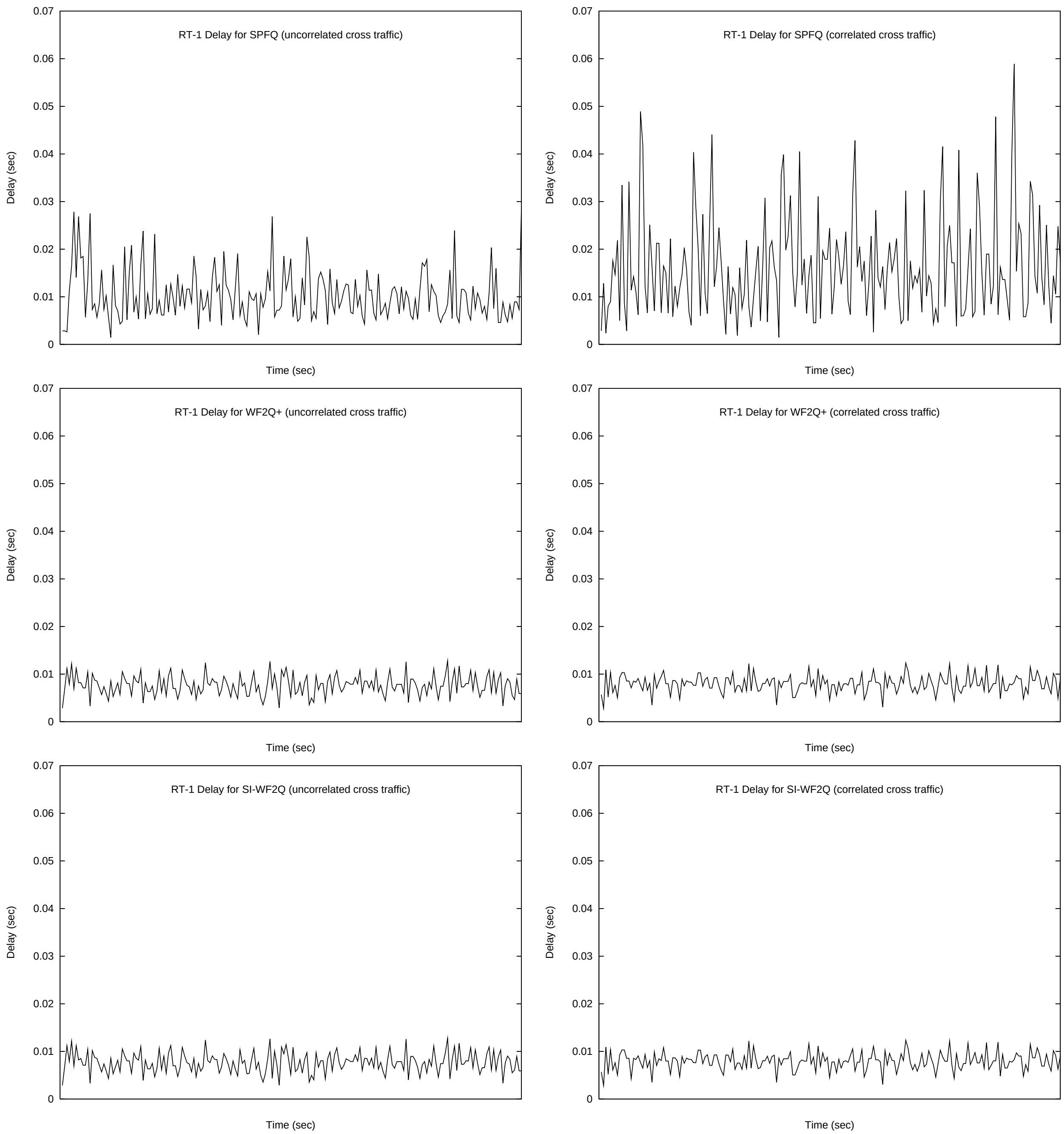

Fig. 7. Delay of RT-1 with Different Schedulers

- Interleaving and stratification of timer wheels allows for controlled but timely processing of timers. This is the key improvement over LFVC.
- The analysis of SI-WF^{2}Q covers the additional error term resulting from an output buffer.
- The paper introduces a new concept of constant execution complexity. Constant complexity is not defined in relation to an arbitrarily sized unit of work. Instead, as long as the execution overhead is strictly proportional to the size of the unit of work (a packet in this case), it is considered constant. Note that the basic complexity consideration is independent of whether the actual execution speed on any particular platform is fast enough to achieve line speed.

In contrast to optimal GPS approximations, such as WF^{2}Q [9] or the improved version in [22], the main limitations of SI-WF^{2}Q are given below. However, these are practical conditions for most realistic configurations.

- A minimum packet size and service rate are required.
- A priority encoder of width $\log_2(\frac{\text{link speed}}{\text{minimum rate}})$ bits is needed. For example, 32 bits can support a range of service rates between 1 Kbit/s and 4 Tbit/s.
- The sum of relative rates must be less than 1.
- Timestamp rounding introduces additional error terms.
- An amortization buffer is required for $O(1)$ complexity.

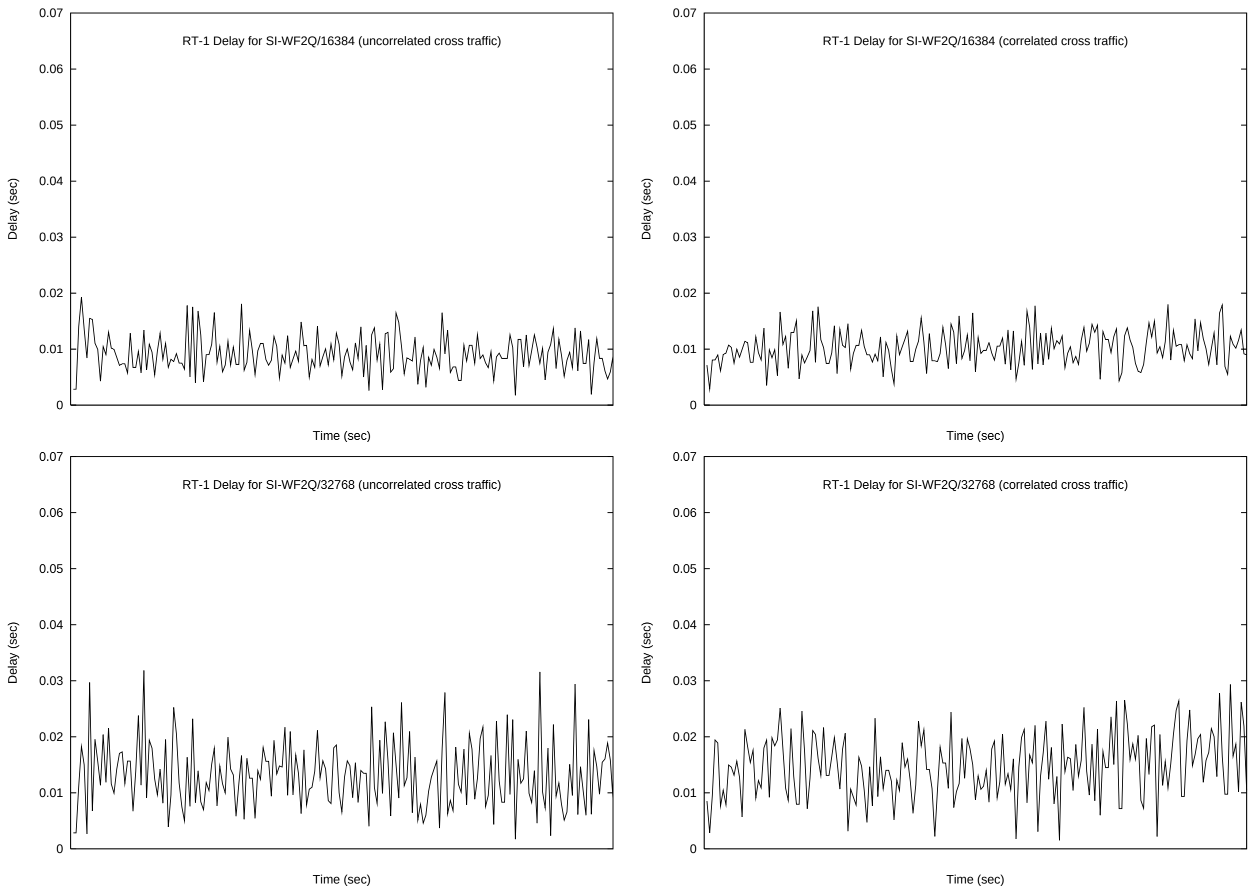

Fig. 8. Delay of RT-1 with Different Settings of λ

VII. Conclusions and Future Work

This paper presents the ISTW data structure for packet scheduling. In conjunction with the given *findNext* and *transferEligible* functions, it enables the construction of the SI-WF^{2}Q packet scheduler that has all desirable properties. The scheduler has low execution overhead and seems very suitable for a low-level implementation sustaining very high packet rates. We are currently in the process of implementing this scheduler on a network processor to verify this assumption.

The bounds shown for SI-WF^{2}Q are not necessarily tight. In particular, it may be possible to find a cheaper alternative to the current start-time search, which can be fully amortized over a shorter time. There is also good reason to believe that just a tighter analysis of the current *findNext* operation on the Low container can already reduce the worst-case amortization buffer.

In order to further reduce the execution overhead, it seems feasible to build a pure start-time scheduler using the ISTW data structure. As a start-time scheduler, it should have good fairness properties. On the other hand, the stratification and interleaving of timer wheels should result in improved delay behaviour over traditional start-time schedulers. Typically, the main delay component for a flow i in a finish-time scheduler is $\frac{L_i}{r_i}$. For such a *Stratified/Interleaved Start-time Fair Queueing* (SI-SFQ) scheduler, we expect this term to grow to $\frac{L}{r_i}$ with the other properties unchanged. However, a detailed investigation of this proposal is a matter of future work.

Both SI-WF^{2}Q and SI-SFQ can be used in regulated mode, that is without work-conservation through start time search and virtual time jumps. In this case, the ISTW data structure provides an efficient technique for real-time traffic regulation.

Last not least, a packet scheduler is only one component in a very complex network architecture. The overall trade-offs of different architecture proposals and service models with respect to the viability of business models, which are mainly shaped by application demand, remain fundamentally unclear. Nevertheless, the availability of an efficient and sophisticated packet scheduler hopefully opens new avenues for the design and operation of packet-switched communication networks.

Errata

If necessary in the future, errata will be available through the author's web page.

Acknowledgements

This work is supported by the Natural Sciences and Engineering Research Council of Canada. Early feedback from S. Keshav has helped shaping its direction. Paolo Valente has pointed out problems in a previous version of the paer and provided valuable feedback. The work in [10] is used extensively to prove SI-WF^{2}Q's properties.

REFERENCES

[1] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[2] S. Shenker, C. Partridge, and R. Guerin, "RFC 2212 - Specification of Guaranteed Service," Sept. 1997.

[3] 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*, vol. 10, no. 4, pp. 529–540, Aug. 2002.

[4] P. Goyal and H. M. Vin, "Generalized Guaranteed Rate Scheduling Algorithms: A Framework," *IEEE/ACM Transactions on Networking*, vol. 5, no. 4, pp. 561–571, Aug. 1997.

[5] 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*, vol. 2, no. 2, pp. 137–150, Apr. 1994.

[6] D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 611–624, Oct. 1998.

[7] S. J. Golestani, "A Self-Clocked Fair Queuing Scheme for Broadband Applications," in *Proceedings of INFOCOM 1994*. IEEE, June 1994, pp. 636–646.

[8] A. K. Parekh, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks," Ph.D. dissertation, Massachusetts Institute of Technology, Feb. 1992.

[9] J. C. R. Bennett and H. Zhang, "WF^{2}Q: Worst-case Fair Weighted Fair Queueing," in *Proceedings of INFOCOM 1996*. IEEE, Mar. 1996, pp. 120–128.

[10] S. Suri, G. Varghese, and G. Chandranmenon, "Leap Forward Virtual Clock: A New Fair Queuing Scheme with Guaranteed Delays and Throughput Fairness," in *Proceedings of INFOCOM 1997*. IEEE, Apr. 1997, pp. 557–565, technical report with full proofs available at http://citeseer.ist.psu.edu/74110.html.

[11] M. Karsten, "SI-WF^{2}Q: WF^{2}Q Approximation with Small Constant Execution Overhead," in *Proceedings of INFOCOM 2006*. IEEE, Apr. 2006.

[12] H. Sariowan, R. L. Cruz, and G. Polyzos, "SCED: A Generalized Scheduling Policy for Guaranteed Quality of Service," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 669–684, Oct. 1999.

[13] I. Stoica, H. Zhang, and T. S. E. Ng, "Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service," *ACM Computer Communication Review*, vol. 27, no. 4, pp. 249–262, Oct. 1997, Proceedings of SIGCOMM 1997.

[14] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *ACM Computer Communication Review*, vol. 19, no. 4, pp. 1–12, Sept. 1989, Proceedings of SIGCOMM 1989.

[15] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks," *ACM Computer Communication Review*, vol. 20, no. 4, pp. 19–29, Sept. 1990, Proceedings of SIGCOMM 1990.

[16] P. Goyal, S. S. Lam, and H. Vin, "Determining End-to-End Delay Bounds in Heterogeneous Networks," in *Network and Operating System Support for Digital Audio and Video, 5th International Workshop, NOSSDAV'95, Durham, New Hampshire, USA*. Springer LNCS 1018, Apr. 1995, pp. 287–298.

[17] D. Stiliadis and A. Varma, "A General Methodology for Designing Efficient Traffic Scheduling and Shaping Algorithms," in *Proceedings of INFOCOM 1997*. IEEE, Apr. 1997, pp. 326–335.

[18] J. C. R. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689, Oct. 1997.

[19] P. van Emde Boas, R. Kass, and E. Zijlstra, "Design and Implementation of an Efficient Priority Queue," *Mathematical Systems Theory*, vol. 10, pp. 99–127, 1977.

[20] G. Varghese and A. Lauck, "Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility," *Operating Systems Review Special Issue: Proceedings of the Eleventh Symposium on Operating Systems Principles, Austin, TX, USA*, vol. 21, no. 5, pp. 25–38, Nov. 1987.

[21] D. Stiliadis and A. Varma, "Efficient Fair-Queueing Algorithms for Packet-Switched Networks," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 175–185, Apr. 1998.

[22] P. Valente, "Exact GPS Simulation with Logarithmic Complexity, and its Application to an Optimally Fair Scheduler," *ACM Computer Communication Review*, vol. 34, no. 4, pp. 269–280, Oct. 2004, Proceedings of SIGCOMM 2004.

[23] Q. Zhao and J. Xu, "On the Computational Complexity of Maintaining GPS Clock in Packet Scheduling," in *Proceedings of INFOCOM 2004*. IEEE, Mar. 2004.

[24] J. Xu and R. J. Lipton, "On Fundamental Tradeoffs between Delay Bounds and Computational Complexity in Packet Scheduling Algorithms," *ACM Computer Communication Review*, vol. 32, no. 4, pp. 279–292, Oct. 2002, Proceedings of SIGCOMM 2002.

[25] D. C. Stephens, J. C. Bennett, and H. Zhang, "Implementing Scheduling Algorithms in High-Speed Networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1145–1158, June 1999.

[26] G. Chuanxiong, "SRR: An O(1) Time Complexity Packet Scheduler for Flows in Multi-Service Packet Networks," *ACM Computer Communication Review*, vol. 31, no. 4, pp. 211–222, Oct. 2001, Proceedings of SIGCOMM 2001.

[27] S. Ramabhadran and J. Pasquale, "Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay," *ACM Computer Communication Review*, vol. 33, no. 4, pp. 239–250, Oct. 2003, Proceedings of SIGCOMM 2003.

[28] X. Yuan and Z. Duan, "FRR: a Proportional and Worst-Case Fair Round Robin Scheduler," in *Proceedings of INFOCOM 2005*. IEEE, Apr. 2005.

[29] B. Caprita, J. Nieh, and W. C. Chan, "Group Round Robin: Improving the Fairness and Complexity of Packet Scheduling," in *Proceedings of the 2005 Symposium on Architecture for Networking and Communications Systems*. ACM Press, Oct. 2005, pp. 29–40.