Implementing Scheduling Algorithms in High-Speed Networks*

Donpaul C. Stephens
Carnegie Mellon University &
Ascend Communications
donpaul@cs.cmu.edu
donpaul@ascend.com

Jon C.R. Bennett
Harvard University &
Xylan Corporation
jcrb@eecs.harvard.edu
jcrb@xylan.com

Hui Zhang
Carnegie Mellon University
hzhang@cs.cmu.edu

Abstract

The fluid Generalized Processor Sharing (GPS) algorithm has desirable properties for integrated services networks and many Packet Fair Queueing (PFQ) algorithms have been proposed to approximate GPS. However, there have been few high speed implementations of PFQ algorithms that can support a large number of sessions with diverse rate requirements and at the same time maintain all the important properties of GPS. The implementation cost of a PFQ algorithm is determined by (1) computation of the system virtual time function and (2) maintaining the relative ordering of the packets via their timestamps (scheduling); and (3) in some algorithms, regulation of packets based on eligibility times. While most of the recently proposed PFQ algorithms reduce the complexity of computing the system virtual time function, the complexity of scheduling and traffic regulation (and therefore the overall complexity of implementing PFQ), is still a function of the number of active sessions. In addition, while reducing the algorithmic or asymptotic complexity has been the focus of most analysis, it is also important to reduce the complexity of basic operations in order for the algorithm to run at high speed. In this paper, we develop techniques to reduce both types of complexities for networks of both fixed and variable size packets. In our approach, regulation and scheduling are implemented in an integrated architecture that can be viewed as logically performing sorting in two dimensions simultaneously. By using a novel grouping architecture, we are able to perform this with an algorithmic complexity *independent* of the number of sessions in the system at the cost of a small *controllable* amount of relative error. To reduce the cost of basic operations, we propose a hardware implementation framework and several novel techniques that reduce the on-chip memory size, off-chip memory bandwidth, and off-chip access latency. The proposed implementation techniques have been incorporated into commercial ATM switch and IP router products.

*The work on ATM Networks was performed while Jon Bennett was at FORE Systems Inc. Donpaul Stephens and Hui Zhang were sponsored by DARPA under contract numbers N66001-96-C-8528 and E30602-97-2-0287, and by NSF under grant numbers Career Award NCR-9624979 and ANI-9814929. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, or the U.S. government.

1 Introduction

Future high-speed integrated-services packet-switched networks will simultaneously support multiple types of services over a single physical infrastructure. In packet switched networks, packets from different sessions belonging to different service and administrative classes, interact with each other when they are multiplexed at the same output link of a switch. The packet service disciplines or the scheduling algorithms at switching nodes play a critical role in controlling the interactions among different traffic streams and different service classes.

Recently, a class of service disciplines called Packet Fair Queueing (PFQ) algorithms have received much attention. PFQ algorithms approximate the idealized Generalized Processor Sharing (GPS) policy [13], which is proven to have two desirable properties: (a) it can guarantee an end-to-end delay to a leaky-bucket-constrained session regardless of the behavior of other sessions; (b) it can ensure instantaneous fair allocation of bandwidth among all backlogged sessions regardless of whether or not their traffic is constrained. The former property is the basis for supporting guaranteed service traffic [13] while the latter property is important for supporting best-effort service traffic [12, 18] and hierarchical link-sharing service [4]. While there are many proposed PFQ algorithms, with different tradeoffs between complexity and accuracy [7, 9, 10, 11, 16, 19, 20, 25], few real implementation exists that can achieve all three of the following goals:

1. support a large number of sessions with diverse bandwidth requirements,
2. operate at very high speeds, 100 Mbps and higher,
3. maintain important properties of GPS (delay bound, fairness, worst-case fairness).

The key difficulty is that PFQ algorithms require buffering on a per session basis and non-trivial service arbitration among all sessions. There are three major costs associated with the arbitration: (1) the computation of the system virtual time function, which is a dynamic measure of the normalized fair amount of service that should be received by each session, and (2) the management of a priority queue to order the transmission of packets; and (3) the management of another priority queue to regulate packets. Weighted Fair Queueing [9, 14], the first proposed and best-known PFQ algorithm, uses the virtual time function defined by the GPS system whose worst case complexity is $O(N)$. More recently, a number of PFQ algorithms have been proposed that have virtual time functions with complexity of $O(1)$ or $O(logN)$ [4, 10, 11, 20]. While the algorithmtic complexity of implementing a priority queue for N arbitrary numbers is $O(logN)$, it is possible to implement Fair Queueing with mechanisms of lower complexity by taking advantage of the properties of PFQ algorithms [16]. However, as discussed in Section 7, there are a number of difficulties in applying this technique in a high speed implementation. In addition, for algorithms that require regulation, one straightforward solution would be to maintain two separate priority queues for regulation and scheduling, but moving packets between these two queues may become a serious bottleneck in high speed implementations.

In this paper, we present a novel architecture that reduces the *overall* complexity of implementing a general class of scheduling algorithms. We begin with the case of networks with fixed length packets, such as ATM. In such a system the server is restricted to supporting a fixed number of rates, and groups together sessions with the same rate. By using the locally bounded timestamp (LBT) property [3], which tightly bounds the difference between the virtual times of sessions with the same rate, it is possible to maintain the priority relationship among sessions in the same group *without* sorting. The problem is then reduced from one that schedules among all sessions to one that schedules among only the sessions that have the smallest timestamp in each rate group. With such an implementation, the complexities for both priority management and virtual time computation grow with the number of discrete rates supported rather than the number of sessions. In addition, the regulation is integrated with scheduling and comes for free. We then extend the grouping architecture to support networks of variable-sized packets. The resulting archiecture can be used to implement not only PFQ algorithms, but also other scheduling algorithms such as those based on service curves [17, 23].

To reduce the complexity of basic operations in hardware implementations, we observe it is important to minimize the bandwidth requirements between on and off chip, improve latency tolerance, and minimize on-chip area. By taking advantage of the globally bounded timestamp (GBT) property [3], which bounds the difference between system virtual time and the virtual start time of all sessions, we can reduce the memory requirements for both systems.

The rest of the paper is organized as follows. In Section 2, we give an overview of GPS and issues in approximating GPS by PFQ algorithms. In Section 3, we describe an architecture that addresses the complexity of both the algorithmic scaling and the basic operations for implementing a class of PFQ algorithms in ATM networks. In Section 4, we extend this architecture to support networks with variable size packets. We present the implementations of these techniques in both ATM networks and packet networks in Section 5.

2 Background: PFQ Algorithms

PFQ are packet approximation algorithms for the GPS discipline [13]. A GPS server has N queues, each associated with a service share. During any time interval when there are exactly M non-empty queues, the server serves the M packets at the head of the queues simultaneously, in proportion to their service shares. All PFQ algorithms use a similar priority queue mechanism based on the notion of a virtual time function. They differ in two aspects: the virtual time function and the packet selection policy.

2.1 System Virtual Time Function

To approximate GPS, a PFQ algorithm maintain a system virtual time $V(\cdot)$ and a virtual start time $S_i(\cdot)$ and a virtual finish time $F_i(\cdot)$ for each session i. $S_i(\cdot)$ and $F_i(\cdot)$ are updated each time session i becomes

active or a packet p_i^k from session i finishes service:

$$S_i(t) = \begin{cases} \max(V(t), F_i(t-)) & \text{session i becomes active} \\ F_i(t-) & p_i^{k-1} \text{ finishes service} \end{cases} \quad (1)$$

$$F_i(t) = S_i(t) + \frac{L_i^k}{r_i} \quad (2)$$

where $F_i(t-)$ is the virtual finish time of session i before the update and L_i^k is the length of the packet at the head of session i's queue.

Intuitively, $V(t)$ is the normalized fair amount of service time that each session *should* have received by time t, and $S_i(t)$ represents the normalized amount of service time that session i has *actually* received by time t. The goal of all PFQ algorithms is to minimize the discrepancies among $S_i(t)$'s and $V(t)$.

The role of the system virtual time function is to reset the value of the session's virtual start time when a previously unbacklogged session becomes backlogged. Different PFQ algorithms use different virtual time functions, which have different tradeoffs between *accuracy* and *complexity*. A virtual time function is accurate if a PFQ algorithm based on it provides service that is almost identical to the GPS algorithm.

Weighted Fair Queueing (WFQ), the best-known PFQ algorithm, and Worst-case-Fair Weighted Fair Queueing (WF^2Q), the most accurate PFQ algorithm, use a virtual time function that is defined with respect to the GPS system. While this is the most accurate virtual time function, it is too complex to implement because its computation involves keeping track of the number and identity of all the active sessions in the *fluid* GPS system, which may change as many times as there are sessions in the system during an *arbitrarily* short time period.

A number of simpler virtual time functions have been proposed that can be calculated directly from the state of the packet system. In the Self Clocked Fair Queueing (SCFQ) algorithm, the virtual time function is defined to be the virtual finish time of the packet currently being serviced, i.e., $V_{SCFQ}(t) = F^{p(t)}$, where $p(t)$ is the packet being serviced at time t. While $V_{SCFQ}(t)$ is quite simple to compute, the resulting SCFQ algorithm provides much larger delay bounds than WFQ. A more accurate virtual time function $V_{FBFQ}(\cdot)$ that can also be computed directly from the packet system was proposed by Stiliadis and Varma [20]. The resulting Frame Based Fair Queueing (FBFQ) algorithm can provide the same delay bound as WFQ.

An even more accurate virtual time function, V_{WF^2Q+} [4], is iteratively defined as follows:

$$V_{WF^2Q+}(t+\tau) = \max(V_{WF^2Q+}(t) + \tau,\ min_{i \in \hat{B}(t+\tau)}(S_i(t+\tau))) \quad (3)$$

where $\hat{B}(t+\tau)$ is the set of sessions backlogged in the WF^2Q+ system at time $t+\tau$.

2.2 Packet Selection Policy

In WFQ, SCFQ, and FBFQ, when the server is picking the next packet to transmit, it chooses, among all the packets in the system, the one with the smallest virtual finish time. We call such a packet selection policy the "Smallest virtual Finish time First" or SFF policy. SFQ uses the "Smallest virtual Start time First"

or SSF policy. Both types of policies use only one of two tags (virtual start and finish times) for packet selection.

While PFQ algorithms with policies using one tag can provide delay bounds almost identical to GPS, they may still introduce large service discrepancies from GPS. In [5], a metric called Worst-case Fair Index (WFI) was introduced to characterize the service discrepancy between a PFQ algorithm and the idealized GPS. It was shown that large WFI's are detrimental to the performance of best-effort and link-sharing services [4, 5]. All PFQ algorithms with policies using one tag (SFF or SSF) have large WFI's.

Two algorithms, WF^2Q [5] and WF^2Q+ [4], were proposed [1] using a different packet selection policy wherein the server chooses, among all the *eligible* packets, the one with the smallest virtual finish time. A packet is eligible if its virtual start time is *no greater than* the current virtual time. Intuitively, a packet becomes eligible only after it has started service in the corresponding fluid system. We call such a packet selection policy the "Smallest Eligible virtual Finish time First" or SEFF policy.

The difference between WF^2Q and WF^2Q+ is that WF^2Q uses a system virtual time function that emulates the progress of the GPS system while WF^2Q+ uses a system virtual function that can be computed directly from the packet system as defined in Eq. (3). It has been shown that WF^2Q is the *optimal* PFQ algorithm in terms of accuracy in approximating GPS. WF^2Q+ maintains all the important properties of WF^2Q: both of them provide the tightest delay bounds and smallest WFI's among all PFQ algorithms. In the rest of the paper, we will use WF^2Q+ as an example to discuss the issues of implementing PFQ algorithms in high speed networks.

2.3 Algorithmic Complexity

Besides the computation of the virtual time function, a second cost of implementing a PFQ algorithm is to maintain a priority queue based on either virtual start time, virtual finish time, or both. Since virtual start and finish times are monotonically increasing within each session, only the first packet of each session needs to be considered when the server selects the next packet to transmit. Thus, the number of entities in the priority queue is the number of active sessions.

Therefore, even though SCFQ, SFQ, and FBFQ have simple virtual time functions, their overall implementation complexity still grows with the number of sessions sharing the link. While it has been demonstrated that sorting can be implemented at very high speed with only several hundred sessions [7], it is unclear whether such implementations can scale to large networks with tens of thousands of sessions competing for a single link. In addition, for algorithms using the SEFF policy such as WF^2Q, WF^2Q+, and EEVDF [22], there is the additional complexity of regulating packets based on eligibility or start times. While the straightforward solution would be to maintain two separate priority queues for regulation and scheduling, moving packets between these two queues may become a serious bottleneck in high speed implementations

[1] We note that WF^2Q+ was independently developed in [21], wherein it was called Start Potential Fair Queueing (SPFQ).

2.4 Implementation Complexity

Most of the research in the literature focuses on reducing the algorithmtic or asymptotic complexity of scheduling algorithms. In high speed implementations, it is also important to reduce the complexity of basic operations[2]. The rapid increase in the performance of silicon chip technology makes it possible to perform on-chip operations across both time (pipelining) and space (superscalar). This level of parallelism is not available for off-chip memory accesses, and this will eventually be the dominating factor as improvements in on-chip performance continue to outpace improvements in off-chip bandwidth. For this reason, off-chip memory bandwidth and latency are the main bottlenecks in high speed implementations. The impact of the high memory latency becomes even more pronounced when there are data dependencies between off-chip accesses, such as when traversing the pointers in a tree-based priority queue.

While on-chip memory is expensive, it provides high bandwidth with low latency; external memory is relatively inexpensive but faces significantly higher latency and bandwidth restrictions. Thus properly partitioning the storage and access requirements between on and off-chip memories is the key to designing cost-effective, high speed, implementations. As we will discuss in Section 5, our architecture can be implemented in an array based structure on-chip with a fixed number of off-chip memory references. This enables us to take advantage of available parallelism on-chip while having a relatively simple component that can be easily designed and verified.

3 Implementing Fair Queueing in ATM Networks

In this section, we present an architecture [3] that can efficiently implement a class of PFQ algorithms in ATM networks which have fixed sized packets refered to as *cells*. In this architecture, the server is restricted to supporting a fixed number of rates and groups together sessions with same rate. The grouping mechanism we employ does not sacrifice the accuracy of the implemented algorithm. For PFQ algorithms with the locally bounded timestamp property Eq. (4), the priority relationship among sessions in the same group can be maintained *without* sorting. The problem is then reduced from one that potentially must schedule among all the sessions in the system to one that only has to schedule among the sessions at the head of each group. With such an implementation, the complexities for both priority management and virtual time computation grow only with the number of discrete rates *simultaneously* supported rather than with the number of sessions.

Furthermore, we note that while available bandwidth to off-chip memory and the latency between sequentially dependent accesses are the main bottlenecks in the implementation of modern computing systems, our architecture adds only two off-chip accesses per cell enqueue or dequeue beyond what is required for a Round-Robin scheduler. The additional off-chip accesses and required storage is exclusively that of a timestamp per session. For PFQ algorithms with the GBT property, we can apply a technique to compress

[2] Arithmetic operations such as $+, -, *, /$ and memory read and writes

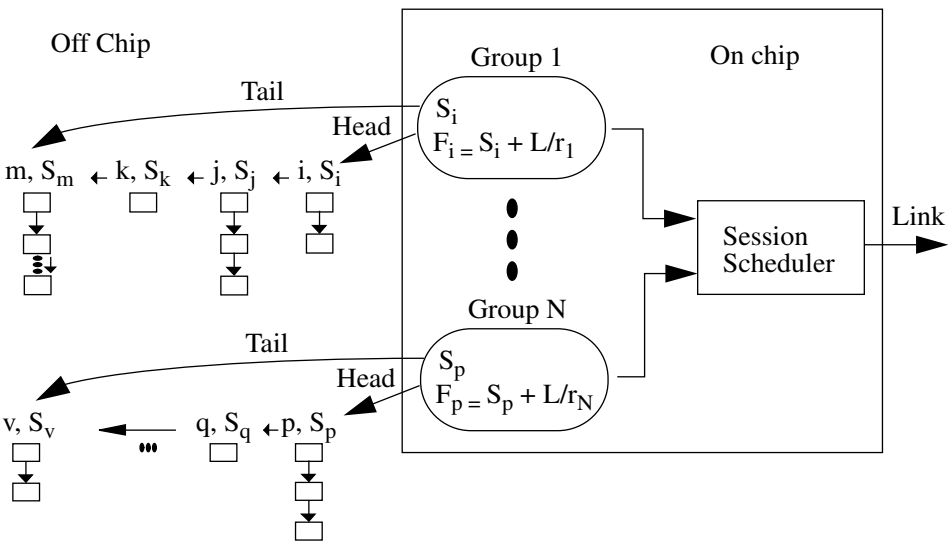

Figure 1: Grouping Architecture

the session timestamps by more than 50%. This reduces both the system cost and the implementation complexity by more than halving both the memory required to store, and the bandwidth required to access, the timestamps. In addition, we use array-based rather than pointer-based data structures on-chip which allows us to take advantage of memory pipelining to greatly reduce the effect of off-chip memory access latency.

Most of the PFQ algorithms in the literature, including WF^2Q+, SCFQ, and SFQ, have both the LBT and the GBT properties, and therefore can be implemented with this architecture.

3.1 Grouping Architecture for ATM Networks

The key difficulty of implementing PFQ algorithms is that the complexity of maintaining the priority queue, and possibly computing the virtual time function, grow as a function of the number of sessions sharing the link. To decouple the implementation complexity from the number of sessions, we introduce the following restriction: at any time, only a fixed number of guaranteed rates are supported by the server. As we discuss in Section 3.3, this restriction will not significantly affect the link utilization. All sessions that share a common rate are associated with a group which stores an entry for each active session. These entries contain a pointer to the head of the session's queue and the virtual starting time for the cell at the head of the session's queue. In each group, the session with the smallest virtual starting time is placed in the scheduler (see Figure 1).

An important advantage of such a grouping architecture is that for any of the three packet selection policies of Smallest Start time First (SSF), the Smallest Finish time First (SFF) or the Smallest Eligible

Finish time First (SEFF), only the packets in the scheduler need to be considered. For algorithms with the SSF policy, this is easy to see as the packet with the smallest virtual start time in the scheduler is also the one with the smallest virtual start time among *all* packets. For algorithms with the SFF policy, we observe that in an ATM network with a fixed packet size, L, for any two sessions i, j that belong to the same group, $(S_i(t) \leq S_j(t)) \Rightarrow (F_i(t) \leq F_j(t))$. This follows directly from $F_i(t) = S_i(t) + \frac{L}{r_i}$, $F_j(t) = S_j(t) + \frac{L}{r_j}$, and $r_i = r_j$. Therefore, the session with the smallest virtual start time in the scheduler is also the one with the smallest virtual *finish* time among *all* packets within the group. For the SEFF scheduling policy we observe that if there are sessions in a group that are eligible, the session within the group in the scheduler must also be eligible as it has the smallest virtual start time in the group. Since, it also has the smallest virtual finishing time of all sessions within the group, the packet with the smallest eligible finish time in the scheduler is also the one with the smallest eligible finish time among *all packets* within the group.

The above grouping architecture reduces the complexity of the scheduler from one that scales with the number of sessions to one that scales with the number of distinct rate groups. It does this without introducing any inaccuracy because it is an *optimization* rather than an *approximation* of the algorithm being implemented. However, there is still a need to select the packet with the smallest virtual start time from each group.

3.2 Maintaining Priority Within Groups

In this section, we show for PFQ algorithms that have the locally bounded timestamp property that, in ATM networks it is possible to maintain a priority queue of sessions for each rate group with a simple linked list.

Definition 1 *Let $Q_i(t)$ be the queue for session i at time t. A PFQ algorithm has the Locally Bounded Timestamp (LBT) property if the following condition holds:*

$$|S_i(t) - S_j(t)| \leq \frac{L}{r_i} \quad \forall i, j \mid \; r_i = r_j, \quad Q_i(t) \neq 0, \quad Q_j(t) \neq 0 \tag{4}$$

Definition 2 *A PFQ algorithm has the Globally Bounded Timestamp (GBT) property if the following condition holds:*

$$S_i(t) - \frac{L}{r_i} \leq V(t) \leq S_i(t) + \frac{L}{r_i} \quad \forall i \;\; s.t. Q_i(t) \neq 0 \tag{5}$$

The properties are so named because Eq. (4) tightly bounds the difference of virtual start times between two sessions in the same rate group, while Eq. (5) which bounds the differences between the system virtual time and virtual start times of all sessions.

WF^{2}Q+ has both the Locally Bounded Timestamp property and the Globally Bounded Timestamp property. Other PFQ algorithms in the literature, including SCFQ and SFQ, also have these properties. We will discuss in Section 5.1 how scheduling algorithms with the globally bounded timestamp property can reduce the memory requirements for their timestamps by 50% or more. For scheduling algorithms with the locally bounded timestamp property, it is possible to maintain a priority queue of sessions in the same

rate group with a simple linked list. Each rate group contains a linked list of the timestamps belonging to the cells at the head of the queue for each session in the group. The entries in the linked list are stored in increasing timestamp order. There are three situations when insertions into the list are needed: (i) the session at the head of the group has finished service, (ii) a new session joins, (iii) and a previously idle session becomes active.

Implementing the exact timestamp assignment as defined by Eq. (1) has two complications. First, since timestamps are represented by finite number of bits n, it is not possible to compare two timestamps that are more than 2^{n-1} time units apart. When a session remains idle long enough, the difference between S_i^k and $V(t)$ can be more than 2^{n-1} time units apart, therefore making it impossible to compute the maximum of the two. When a session i arrives into an empty queue, we can employ Eq. (1) and Eq. (5) to bound the session's virtual start time to be within the range of valid times, $[V(t), V(t)+\frac{L}{r_i}]$. Sessions whose timestamps lie outside of this range should ideally be reset according to Eq. (1). However, sessions that have been idle for an integer number of overflows of the system virtual time could also appear within this range. For this reason, when representing timestamps with a finite number of bits, the session start time could be assigned a value between $[V(t), V(t)+\frac{L}{r_i}]$. A second complication is that in order to maintain the sorted order of the group, an arriving session has to be inserted into the correct relative position within the group.

Q_i	Queue of cells for session i
G^{head}	Session at the head of the group G queue
r_i	Service rate for session i
G_i	Rate group for session i
$\mathcal{B}_G(t)$	Set of backlogged groups at time t
$\mathcal{E}_G(t)$	Set of eligible groups at time t, $\forall\ G \in \mathcal{B}_G(t) \| S_{G^{head}} \leq V(t)$

SCHEDULE-CELL(t)

```
1   /* Find group containing session to be transmitted */
2   for ∀J ∈ E_G(t)
3       do if F_{J^head} ≤ F_{G^head}
4             then G ← J
5   i ← DEQUEUE-SESSION(G)
6   TRANSMIT-CELL(Q_i)
7   S_i ← S_i + L/r_i
8   STORE(S_i)
9   if Q_i ≠ ∅
10     then ENQUEUE-SESSION(G, i)
11  LOAD(S_{G^head}) /* For new head session */
```

When the session at the head of the linked list i is served it will have its new starting time computed as $S_i = S_i + \frac{L}{r_i}$. From Eq. (4), we know that this new starting time will be larger than the timestamp of any other session in its rate group. Therefore, simply moving the session to the tail of the linked list will maintain the sorted order of the list.

CELL-ARRIVAL(c, i)

```
 1  if Q_i = ∅
 2     then  /* Session was idle */
 3           /* Protect against timestamp wraparound */
 4           LOAD(S_i)
 5           S_i ← MIN(V(t) + L/r_i, MAX(S_i, V(t)))
 6           if G_i^head ≠ ∅
 7              then  /* Maintain ordering of timestamps */
 8                    S_i ← MAX(S_i, S_{G_i^tail})
 9           STORE(S_i)
10           ENQUEUE-SESSION(G_i, i)
11  ENQUEUE-CELL(Q_i, c)
```

When a previously idle session i becomes backlogged at time t, we append the session to the end of the list. When a session i arrives into an empty queue, its virtual start time will be within the range $[V(t), V(t) + \frac{L}{r_i}]$. The lower bound is given by Eq. (1) and its upper bound by Eq. (5). If the rate group that session i belongs to has other backlogged sessions there is need to preserve the locally bounded timestamp property and the sorted order. To do this we insure that S_i is at least as large as the timestamp of the last session in G_i while still conforming to the range $[V(t), V(t) + \frac{L}{r_i}]$ The effects of this operation on the delay bound is explained in Section 6.

Compared to a Round Robin scheduler, our scheduler adds only two additional off chip accesses when a group is scheduled (procedure Schedule-Cell lines 8 and 11), and two accesses when a cell arrives to an empty session queue (procedure Cell-Arrival line 4 and 9).

3.3 Implications of Limited Rates

One issue is how many rates need to be supported and how the restriction of supporting only a fixed number of rates affect the network utilization. It has been demonstrated by Charny [8] that in the context of the ABR service, high network utilization can be achieved even with differences as large as a 10% between consecutive rates. By assigning the group rates by such an exponential distribution, a wide range of rates can be supported with a relatively small number of groups. It should be also noted that the number of rates that can be used at any time is strictly less that the number of different rates that may exist in the system. For example, a system can only have one session active at any time with a rate between one half of a link's

rate and the link rate, regardless of granularity.

For any given link rate and granularity, we can determine the maximum number of rates that can be used at the same time by summing the possible rates starting with the smallest until the link capacity is met. For example, a 150 Mbps link that supports rates in integer multiples of 10 Mbps, $(10, 20, 30, \cdots, 140, 150)$ has 15 different possible rates. However, no more than 5 may be active at one time because $10+20+30+40+50 = 150$. It is also shown in [8] that it is necessary for networks to support at least a common subset of the possible rates in order to insure interoperability of equipment. As a result we do not believe that there is anything unreasonable or limiting in these assumptions.

4 Regulation and Scheduling in Packet Networks

The techniques presented in Section 3 enable the efficient implementation of WF^2Q+ and its rate controlled variant in ATM networks that have fixed packet sizes. However, they are not applicable to variable-packet-size networks such as Ethernet, Frame Relay, and Packet Over SONET. In this section, we present techniques to efficiently implement scheduling algorithms in networks with variable packet sizes.

4.1 Generalized Grouping Architecture for Packet Networks

As discussed in 3, one of the key advantages of the the ATM grouping architecture is that it reduces overall complexity of priority queue management from the number of sessions to the number of groups. A second important advantage of the grouping archiecture is that it performs regulation (based on virtual start times S_i) and scheduling (based on virtual finish times F_i) in an integrated manner by taking advantage of the fact that the S_i and F_i are related ($F_i = S_i + \frac{L}{r_i}$), and therefore reduces the worst-case complexity of the overall system.

In this section, we will present a generalized grouping architecture that is applicable to packet networks while maintaining most of the advantages of the original grouping architecture. In order to perform regulation and scheduling among tens of thousands of queues in a packet system, we again need to employ an architecture that performs both operations in an integrated manner. Instead of moving between two successive one dimensional priority queues, we propose a two dimensional sorting structure in which the sorting keys are $F_i - S_i$ and F_i respectively. We call $F_i - S_i$ the service interval of session i, denoted by Φ_i. As in the ATM system which uses a discrete set of rates to group sessions, we will use a discrete set of service intervals to group sessions.

Within each group, the system will employ a designated timestamp (either S_i or F_i) for sorting among sessions within the group. For most scheduling algorithms, it is preferable that F_i be used for sorting within the groups, as this enables tighter delay bounds to be achieved by the implemented algorithm. S_i will be inferred from F_i and the service interval of the group the session belongs to. Only the session with the smallest F_i within each group will be considered for scheduling between groups. As with the ATM

implementation, this packet implementation need only to perform a scan amongst the G groups.

While these conceptual features remain, the variable sized nature of packets imposes key operational restrictions that require major changes in the implementation techniques. In particular, a session may not always be bound to the same group because the session's service interval is dependent upon not only its scheduling parameters (e.g. rate) but also the length of the first packet in the session's queue. This has two consequences. First, there will be a large number of possible values of session service intervals and only a small number of service interval groups. To achieve a desired level of accuracy in the approximated algorithm, we need to choose the proper number and values of the service intervals that depend on both the supported ranges of rates and packet lengths. In addition, care needs to be taken when a session is mapped to a group so that the inaccuracy introduced by the approximation does not significantly affect the performance guarantees that can be provided by the approximated algorithm. Secondly, while a session's queue will always stay in the same group in the ATM scheduler, it may change groups in the packet scheduler when a new packet comes to the head of the queue as the service interval may change due to the change of the packet length. When a session changes groups, it can be inserted at an arbitary position in the priority queue of the new group. Therefore, it is not longer possible to maintain a sorted relationship within each group with only a simple FIFO queue. We discuss these two issues in the next two sections.

4.2 Discretization of Service Intervals

Similar to the case of ATM while a session needs to be put in the closest group with a group rate not less than the session's rate, care needs to be taken when a session's service interval is discretized. In particular, we will round *up* the session's service interval Φ_i to the smallest group service interval Φ_g that is not less than Φ_i, i.e.

$$\Phi_g \geq \Phi_i^k = F_i^k - S_i^k > \Phi_{g-1} \tag{6}$$

Which we use to derive the approximated virtual start time $\widehat{S}_i$ used for regulation:

$$\widehat{S}_i^k = F_i^k - \Phi_g^k \tag{7}$$

Which, in turn is bounded by:

$$S_i^k - \beta_i^{max} < \widehat{S}_i^k \leq S_i^k \quad \forall i,k \tag{8}$$

Where β_i^{max} represents the maximal amount of virtual time that a session i can appear eligible earlier in the approximate system than in the non-approximate system.

$$\beta_i^{max} = \max(\Phi_g^k - \Phi_i^k) \quad \forall i,k \tag{9}$$

Since derived virtual start time $\widehat{S}_i$ is computed by Eq. (7), Eq. (6) ensures that it is not larger than the actual virtual start time. Recall that in WF^{2}Q+, the virtual start times of the sessions are used for two purposes: (a) to determine if the session is currently eligible for transmission, and (b) to advance the

system virtual time. By rounding up the length of the session's service interval which results in a smaller approximated virtual start time $\widehat{S}_i(t)$, the first packet in the session can become eligible *earlier* and the system virtual time may grow *slower* during a short period in the approximated system than in the original system. As will be discussed in Section 6, this should not affect the worst-case delay bound that can be provided by the approximated system. Alternatively, if the length of the session's service interval were rounded down, the derived virtual start time could be later than the actual start time and the system virtual time could advance faster than it would in a non-approximated system, which would in turn affect the achievable delay bounds.

As in the case of the ATM implementation where the group rates need to be discretized, we need to discretize the service intervals in the packet implementation. The key difference is that while the number and values of rates depends solely on the supported range of rates in the ATM system, the number and values of service intervals in the packet system depend on both the supported ranges of rates and packet lengths.

Provided that the entire range of packet sizes from the media's minimum to maximum packet size is permitted for all rates, the maximum and minimum service intervals that can be present are given by the following:

$$\Phi_{min} = \frac{L_{min}}{r_{max}} \tag{10}$$

$$\Phi_{max} = \frac{L_{max}}{r_{min}} \tag{11}$$

For some number of groups G to span this range with a given granularity Δ, the following relationship needs to be satisfied:

$$\Delta^{G-1} = \frac{\Phi_{max}}{\Phi_{min}} = \frac{L_{max} * r_{max}}{L_{min} * r_{min}} \tag{12}$$

With such a uniformly exponential spanning of the groups, a session i with service interval Φ_i would be placed into the group that satisfies the following equation:

$$g = \left\lfloor log_{\Delta} \frac{\Phi_i^k}{\Phi_{MIN}} \right\rfloor + 1 \tag{13}$$

To put this into context, consider a 1 Gbps link in a network with packet sizes ranging from 64 to 1500 Bytes in length. Furthermore, it is desired that the network support any session rate from 64 Kbps to 100 Mbps. This results in a range of service intervals from 5.3 to 195,312 packet per second (pps), which is a factor of 39,000. This range may be spanned with a granularity Δ of 2, 1.25, and 1.18 between successive groups with 16, 48, and 64 groups respectively. While the tradeoff in the delta to the number of groups is the same as in the ATM scheduler, the implications of the delta are different. In the ATM system, the number of groups define the rates that can be assigned to the sessions. In the packet system, the number of

groups defines the accuracy with which a session's service interval could be expressed during a single packet scheduling operation. Note that *any* session rate within the given range may be supported.

4.3 Sorting within groups

While a session queue will always stay in the same group in the ATM scheduler, it may change groups in the packet scheduler when a new packet comes to the head of the queue as the service interval may change due to the change of the packet length. When a session changes groups, it can be inserted at an arbitary position in the priority queue of the new group. Therefore, in the packet system, it is not longer possible to maintain a sorted relationship within each group with only a simple FIFO queue, instead an explicit priority queue mechanism must be used. However, since the task is to sort the virtual finish times of flows that have similar service intervals, it can be shown that the virtual finish times of sessions in the group span a range of $\frac{L_{max}}{L_{min}}$ times the service interval. Therefore, if we are willing to tolerate an increase of the delay bound by a fraction of one service interval (like the case in the ATM implementation), we can reduce the complexity of sorting by measuring the virtual finish times in units of fractions of the group's service interval.

For example, for the network mentioned above, the mechanism would thus need to support a range of $\frac{L_{max}}{L_{min}} = \frac{1500}{64} = 23.4$ times each group's service interval. This can be easily accomplished by using a calendar queue mechanism with 32 entries. To further increase the accuracy, each calendar entry can be further divided into sub-interval "bins". For example, to achieve an accuracy of 50%, 25%, and 6% within this range, 2, 4, and 16 sub-interval "bins" need to be allotted per interval Φ respectively. Combined, we have a two-level hierarchical calendar queue (also called a trie). In this a structure, a bit will be set in a higher level unit to imply that a bit is set somewhere in the word beneath it. In Figure 2, these are the marked locations whose sub-unit is expanded. Searching at each level can be easily implemented by a scan operation.

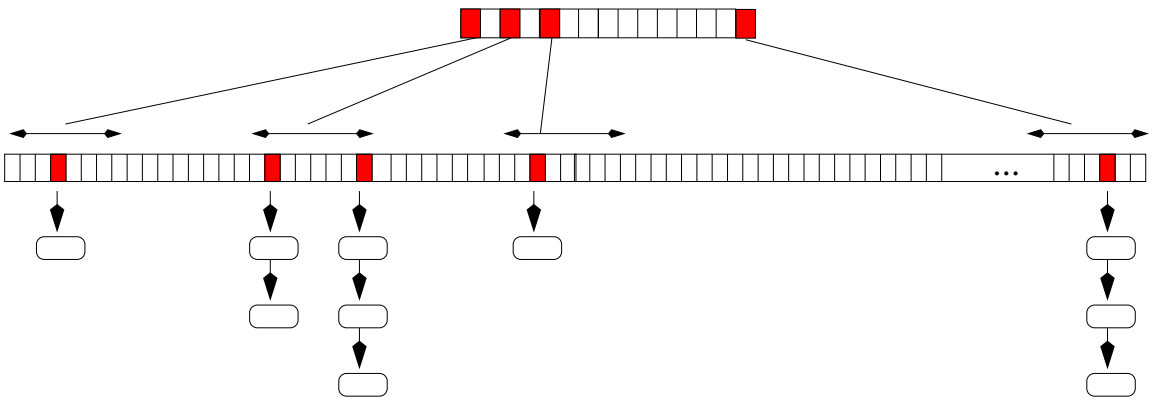

Figure 2: Hierarchical Calendar Queue for Intra-Group Scheduling

Note that placing sessions into the sorting structure, the approximate virtual finish time used for scheduling a packet must not be less than the computed virtual finish time. If this were not so, sessions could miss

their deadlines due to the server erroneously sending packets with later deadlines. We define the approximate virtual finish time that is used by the server by:

$$F_i(t) \leq \widehat{F}_i(t) < F_i(t) + \gamma_i^{max} \quad \forall i, t \tag{14}$$

where γ_i^{max} represents the maximal amount of virtual time that a session i's virtual finish time can appear to be later in the approximated system than in the non-approximated system. Thus, a session is scheduled as if its $\widehat{F}_i(t)$ is the largest value in the bin (all lower bits are set). The $\widehat{S}_i(t)$ is computed by subtracting the group's service interval from the smallest value in the bin (all lower bits are cleared).

The worst case complexity is as follows: an enqueue requires one insertion into a linked list along the leaves of the trie and potentially one replacement of the value within the group data structure. A dequeue requires one scheduler selection among the elements within the group data structure, one removal of a session from the head of a linked list, plus the cost of an enqueue. Note that the complexity is *not* a function of the number of sessions. We find it interesting to note that the on-chip complexity of the packet scheduler is actually simpler than that of the ATM scheduler.

5 Hardware Implementation

5.1 Implementing Timestamps

The size of the timestamps used determines both the range of supportable rates and the accuracy with which those rates may be specified. In addition, they determine the scheduler's memory requirements in terms of bandwidth and storage space, both on-chip and off-chip. In this section we will show that for any service policy which maintains the *globally* bounded timestamp property that the size of the timestamps in the system need only be 1 bit larger than the number of bits needed to represent the service interval of the *largest* rate in the system, which is the service interval which requires the *fewest* bits to represent. Before discussing the details of how this is accomplished we will first give some examples of how comparisons are performed in modular arithmetic to help provide some intuition into our approach. We will then discuss the details of how we represent service intervals and the units in which we measure time. Finally we will show how these concepts come together to allow us to greatly reduce the size of our timestamps without any appreciable impact on their accuracy.

5.1.1 Comparisons in modular arithmetic

Using modular arithmetic, timestamps represented by finite number of bits n can be compared without ambiguity if difference between them is is less than 2^{n-1}. Using this property, Rexford et al [15] suggested that the size of the timestamps in the system only need to be a few bits larger than the number of bits needed to represent the smallest rate in the system, r_{min}, as it has the largest service interval, $\Phi_{max} = \frac{L}{r_{min}}$.

However in a system where some sessions have very small rates, a relatively large number of bits may still be needed to represent the timestamps.

For scheduling algorithms that have a tight bound between the timestamps of the sessions and the system virtual time, we can exploit this relationship to further reduce the number of bits needed to represent the timestamps. In the case of WF^{2}Q+ in ATM networks, the Globally Bounded Timestamp (GBT) property (5) bounds the maximum discrepancy between the system virtual time $V(t)$ and the virtual start times $S_i(t)$ of any session i to one service interval Φ_i. Due to the tightness of the bounds in this system, we can successfully compare the system virtual time with the timestamp of a high rate session using only one more bit than is needed to represent the service interval of that session, as the higher order bits may be inferred from the bound.

Using the notation $X[j:i]$ to represent the binary number formed from the bits in X in the range j down to i, a modular arithmetic $A > B$ comparison can be computed as follows:

GT(A, B, n)
1 **if** $A[n-1:0] > B[n-1:0]$
2 **then** $result = TRUE$;
3 **else** $result = FALSE$;
4 **if** $A[n] = B[n]$
5 **then return** $result$
6 **else return** $NOTresult$

For example, consider the following 4 bit numbers: $A = 1110_2$ and $B = 0001_2$. Without considering rollover, it would appear that $A > B$. However, because the difference between them cannot exceed 8, we note that $B = A + 3 \Rightarrow A < B$. A more complete explanation of how to compare numbers in the presence of wraparound can be found in the discussion of TCP (Transmission Control Prototcol) sequence numbers by Wright and Stevens [24].

5.1.2 Representing Time

In an ATM system it may seem tempting to use an integer representation of the virtual time in units of seconds per cell, so that the virtual time is simply incremented by 1 each time a cell is sent. However, this would result in service intervals which are integer multiples of a cell time, adversely affecting the provisioning of high rate sessions. In a system with an integer virtual time the fastest service intervals would be every 1, 2, 3, .. cell times, which would in turn represent only rates of 1, $\frac{1}{2}$, $\frac{1}{3}$, .. times the link rate. As a result, session rates between $\frac{1}{2}$ and $\frac{1}{3}$ of the link rate could not be represented. Another problem is that the rate of increase of the virtual time function is not guaranteed to be integer valued. To avoid these problems, we choose to represent virtual time as an N bit fixed point number with $N - M$ bits in the integer portion and M bits in the fractional portion. Figure 3 shows $V(t) = 4294966784_{10}$ in fixed point format with $N = 32$ and $M = 9$ with the service intervals for a high rate session i, Φ_i, and a lower rate session j, Φ_j.

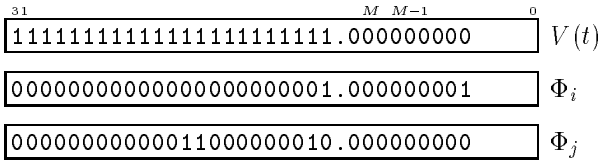

Figure 3: Fixed point representation of $V(t)$, Φ_i and Φ_j

In the fixed point system we have chosen, Φ_i corresponds to the second largest rate which can be represented, equal to $\approx .99805$ times the link rate. Assume that we have chosen the number of bits in the fractional portion of our timestamp so that the delta between the two smallest representable service intervals, in this case $\approx .195\%$, is acceptably small. It would then seem reasonable to accept this same relative delta for larger service intervals. This then leads us to storing the service intervals as floating-point numbers with M bits of mantissa and $\log_2(N) = 5$ bits of exponent. Figure 4 shows the service interval for sessions i and j in floating-point format. Note that we use a normalized floating-point format and thus do not store the leading 1 of the mantissa, m_i.

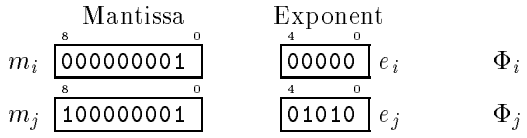

Figure 4: Floating-point format for Φ_i and Φ_j

This interval compression technique is not specific to our timestamp techniques and is genericly applicable to *any* PFQ algorithm.

5.1.3 Compressed Timestamps

Having introduced the notion that we can accept the same degree of *relative* accuracy in the representation of service intervals for different sessions, we now introduce the same concept for the representation of timestamps. Figure 5 shows the computation of S_j by the addition of $V(t)$ and Φ_j. In order to perform the addition of Φ_j we first left shift the mantissa m_j by e_j bits and then zero fill the low order bits. Based on the

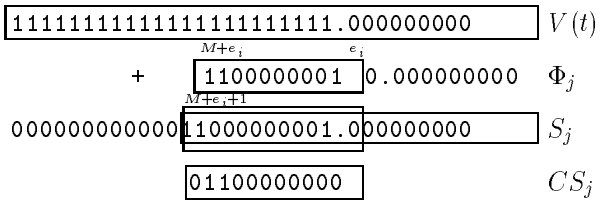

Figure 5: Computing the value of CS_j

discussion in Section 5.1.1 we can successfully compare $S_j[M+e_j+1:0]$ against $V(t)$. If instead of storing $S_j[M+e_j+1:0]$, we choose to store $S_j[M+e_j+1:e_j]$, shown by the wider box around S_j in Figure 5 we can reconstruct S_j in the much the same manner that we constructed the fixed point representation of Φ_j from it's floating-point representation. In order to avoid some problems which could be caused by truncating the low order bits, instead of zero-filling the low order bits we one-fill the low order bits. In the worst case this would be equivalent to having an interval which is almost as large as the next smaller representable rate. Since $S_j[M+e_j+1:e_j]$ has the same number of bits as $S_i[M+1:0]$, it represents S_j with the same *relative* accuracy with which $S_i[M+1:0]$ represents S_i.

5.1.4 Reconstructing Full Length Timestamps

The previous section demonstrated that it is possible to compress a 32 bit timestamp down to a much smaller number of bits and then uncompress the number so that it could be compared against the system virtual time, $V(t)$. When comparing a session's timestamp against $V(t)$ to determine if session i is eligible we only needed to reconstruct the lower $M+e_i+2$ bits of S_i. To compare session i's finish time against session j's finish time, where $e_j > e_i$, we will need to reconstruct the lower $M+e_j+2$ of S_i as well as S_j. In the case where session j is the lowest rate session in the system this amounts to having to reconstruct all 32 bits of S_i. The method by which we reconstruct the full length timestamp depends on whether the upper bit of the compressed timestamp is the same as the corresponding bit of the current value of $V(t)$. If the upper bit of the compressed time stamp is the same as the corresponding bit of $V(t)$, then the upper bits of the full length timestamp are the same as the upper bits of $V(t)$. If the upper bit of the compressed timestamp is different than the corresponding bit of $V(t)$ then there has been a rollover of some of the upper bits of the timestamp. To reverse the rollover, we simply add or subtract a 1 from $V(t)$ in the $M+e_1+2$'th bit position depending on whether the $V(t)$ is respectively less than or greater than S_i. Below is the pseudocode.

RECONSTRUCT-TIMESTAMP$(V(t), CS_i, e_i)$

1 $M' \leftarrow M + e_i + 1;$
2 $S_i[M' : e_i] \leftarrow CS_i;$
3 **if** $e_i > 0$
4 **then** $S_i[e_i - 1 : 0] \leftarrow 1 \cdots 1;$
5 **if** $V(t)[M'] = S_i[M']$
6 **then** $S_i[31 : M'+1] \leftarrow V(t)[31 : M'+1];$
7 **return** $(S_i);$
8 **if** $\mathrm{GT}(V(t), S_i, M'+1) = TRUE$
9 **then**$S_i[31 : M'+1] \leftarrow V(t)[31 : M'+1]-1;$
10 **else** $S_i[31 : M'+1] \leftarrow V(t)[31 : M'+1]+1;$
11 **return** $(S_i);$

5.2 ATM Networks

Using the methods outlined in Section 3, we have implemented a controller for a FORE Systems ATM switch 8 port network module. This controller implements a rate controlled variation [2, 26] of WF^2Q+ [1] to address the requirements of the service provider marketplace. It should be noted that a work conserving WF^2Q+ scheduler could be implemented with the same complexity as the rate controlled variant.

The implementation provides each port with up to 64 different rate groups. There is only one instance of the scheduling hardware which is shared among 8 ports. The scheduler is completely self-contained and can schedule an outgoing VC in about 500ns. The implementation was done in 3V 0.5um 3LM CMOS technology. The scheduler part of the chip operates at 60MHz and requires only 15K gates. To support all types of service, beyond just those requiring tight shaping, there are two statically prioritized round-robin queues into which UBR (Unspecified Bit Rate), ABR (Available Bit Rate) and VBR (Variable Bit Rate) sessions can be queued. It is important to point out that we can enqueue a VC into one of the round-robin queues even if it has already be placed into the rate controlled WF^2Q+ scheduler. So ABR and VBR VCs can be placed into both servers, where the rate controller will provide any minimum bandwidth guarantees, and the round-robin will allocate the excess bandwidth.

The exact implementation consists of two symmetric blocks in the scheduler: each one searches half of the 64 groups. In each block, there is a memory in which the start times of 32 of the 64 groups are stored. 256 entries are needed for the 8-port configuration. There is also a time interval table which indicates what rate has been assigned to each of the 64 groups. Each block finds the group with the smallest finish time among all eligible groups through linear search in 32 60MHz cycles. After 32 cycles, the eligible group with the smallest finishing time in each memory is found. The top level block will pick the final minimum elements among two blocks and schedule the head VC of the group. If there is no eligible group, or the VC at the head of the queue for the selected group has no cells, then the scheduler will service a VC from the round-robin, if one exists. Since this can all be done in 500ns, the scheduler is fully capable of OC-12c (622Mbps) operation.

5.3 Packet Networks

Using the methods outlined in Section 4, we have implemented a linecard controller for a router prototype at Ascend Communications. The controller maintains all queue counters, buffer management, and a scheduling unit that can operate as either in a work conserving or non-work conserving (rate controlled) WF^2Q+ mode.

The controller maintains 16 groups per-port, with a uniform Δ of 2 between the service intervals of successive groups. Having a Δ of 2 dramatically simpifies the computation of the group number (Eq. 13), as the log_2 function reduces to a search for the highest bit set in the session's Φ_i. For our choice of system parameters, we cover a range of $128 * \Phi_g$ in virtual timespace for each group wherein we have 4 sub-bins per Φ_g (thus, $\gamma_g = 0.25 * \Phi_g$) to enable more precise placement of the $F(t)$. In order to facilitate constant time insert and delete-min operations, we employ a two-level trie with 32 entries in the top level and 16

subinterval bins associated with each first level entry. This structure can efficiently be searched for the first non-empty subinterval bin, relative to the current virtual time's logical position in the structure. To reduce memory bandwidth, the first level of the trie is cached on-chip along with the first bin with data (which also happens to be the timestamp).

The server selects the next packet to send by scanning all of the groups, looking at only the (approximated) timestamps for the flows in each group with the smallest $\widehat{S}_i(t)$, $\widehat{F}_i(t)$ pair. Because the group service intervals are fixed, we embed the group's service interval into the logic that compares the virtual time to the group's subinterval bin (which, as stated earlier is its earliest flow's timestamp). The resulting packet system can be constructed out of very simple hardware circuits, with the most complex component being the multiplication function required to compute $\frac{L}{r_i}$ (which is required for *any* PFQ algorithm). These functions are capable of operating at over 1 Mpps, which is sufficient to support two OC-12c POS media interfaces. With current ASIC technology, we believe these techniques could easily be applied to support speeds sufficent for OC-48c media interfaces.

6 Delay Results for Approximated Scheduling

Theorem 1 *A cell of session i in the ATM scheduling architecture presented in Section 3 will miss its deadline by no more than $\tau + \Phi_i$, where τ represents the time to transmit an ATM cell on the link and Φ_i is the service interval of session i.*

The proof may be found in Appendix A.

Theorem 2 *A packet of session j will miss its deadline by no more than $\tau + \beta_j^{max} + \gamma_j^{max}$, where τ represents the time to transmit a maximum sized packet on the link, β_j^{max} represents the maximum amount the session may be deemed by the server to be eligible before its computed virtual start time, and γ_j^{max} represents the maximum amount the server may inflate the session's virtual finish time.*

The proof may be found in Appendix B.

We conjecture that as long as $\widehat{S}_i(t) \leq S_i(t)$, there is no effect on the session's delay bound. Thus, a packet of session j will miss its deadline by no more than $\tau + \gamma_j^{max}$.

7 Related Work

For ATM networks an idea for scheduling among rate groups similar the one used in the commercial $OC-12c$ implementation [6] we presented was later independently developed by Rexford et. al who were the first to present it in the literature [15]. With Rexford's algorithm, sessions with similar throughput parameters are placed into one of a small number of groups. The server scheduled among the groups which then provided service to sessions, as if they were nodes in a hierarchical scheduler. Fundamentally this mechanism is

a heuristic approximation of the desired scheduling algorithm. As a result, in the worst case it cannot provide minimum bandwidth guarantees due to the variation in the set of sessions between the packet and corresponding fluid systems, as briefly discussed in [15].

In contrast, our grouping techniques are solely used as an optimization to simplify the sorting for both regulation and scheduling of the sessions themselves. This enables our mechanism to retain the fairness properties of the implemented algorithm while incurring only a small increase in the session's delay bound.

By observing that the range of virtual times of all sessions at any given time is bounded, Suri, Varghese, and Chandranmemon map the priority queue management problem to that of searching in a finite-universe [16]. In such a universe, a priority queue can be used that supports insert, delete, and successor in $O(loglogN)$ time, when the elements are in the range $[0, N]$. This is a particularly attractive solution for algorithms with SEFF policy (such as LFVC and WF^{2}Q+) while there is a tight bound among the virtual times of all sessions, i.e., the range N is quite small. However, the $O(loglogN)$ result holds only for the priority queue based on virtual finish times, there is still the problem arising from the interaction of the two priority queues. In particular, whenever the server is selecting the next packet for service, it needs first to move all the eligible packets from the priority queue based on eligibility times to the priority queued on virtual finish times. In the worst case, all N packets must be moved between the two priority queues before selecting the next session for service.

8 Conclusion

In this paper, we develop techniques to reduce both the asymptotic and basic operation complexities of implementing scheduling algorithms in high speed networks. For fair queueing algorithms with the locally bounded timestamp property, we propose a grouping mechanism that reduces the complexity of sorting so that it grows as a function of the number of distinct rates in the system. To reduce the cost of basic operations, we propose a hardware implementation framework and several novel techniques that reduce the on-chip memory size, off-chip memory bandwidth, and off-chip access latency. In particular, for service policies with the globally bounded timestamp property, we present a technique that compresses the size of the timestamps, which have to be accessed from off-chip memory during each cell time, by 50% or more. These techniques introduces little inaccuracy for the implemented algorithms and may be used for any scheduling algorithm for which these properties hold, including SCFQ, SFQ, and WF^{2}Q+. The resulting system implements Fair Queueing in ATM systems with no external memory required outside that used for storing a single timestamp value, and that in a very compact form. This forms the basis for a commercial $OC-12c$ ATM hardware implementation.

We then extend our conceptual framework to address a more general class of scheduling algorithms for networks with variable sized packets. For these systems, we discretize not the set of usable rates for external traffic, but rather the the service intervals used for internal scheduling. Regulation and scheduling are implemented in an integrated architecture that can be viewed as logically performing sorting on two

dimensions simultaneously. The packet system achieves further on-chip optimizations by hard-coding the mapping between the groups and their intervals. The resulting system, while requiring additional external memory, can implement a broader class of scheduling algorithms, including the FSC algorithm, for networks with variable sized packets.

9 Acknowledgement

We are very grateful to Ion Stoica for helping with the proofs used in the paper. We would also like to thank Jennifer Rexford, Anna Charny, Mike Smith, and anonymous reviewers for their insightful comments and suggestions.

References

[1] J.C.R. Bennett. Method and a scheduler for controlling when a server provides service to an entity, October 1998. United States Patent 5,828,879.

[2] J.C.R. Bennett. Method and a scheduler for controlling when a server provides service with rate control to an entity, December 1998. United States Patent 5,845,115.

[3] J.C.R. Bennett, D.C. Stephens, and H. Zhang. High speed, scalable, and accurate implementation of packet fair queueing algorithms in ATM networks. In *Proceedings of IEEE ICNP '97*, pages 7–14, Atlanta, GA, October 1997.

[4] J.C.R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of the ACM-SIGCOMM 96*, pages 143–156, Palo Alto, CA, August 1996.

[5] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case Fair Weighted Fair Queueing. In *Proceedings of IEEE INFOCOM'96*, pages 120–128, San Francisco, CA, March 1996.

[6] J.C.R. Bennett, F. Zhou, R.J. Brownhill, and M.N. Ganmukhi. Digital network including mechanism for grouping virtual message transfer paths having similar transfer service rates to facilitate efficient scheduling of transfers thereover, October 1997. EU Patent EP0800296.

[7] H. Chao. Architecture design for regulating and scheduling user's traffic in ATM networks. In *Proceedings of ACM SIGCOMM'92*, pages 77–87, Baltimore, Maryland, August 1992.

[8] A. Charny, K.K. Ramakrishnan, and A. G. Lauck. Scalability issues for distributed explicit rate allocation in ATM networks. In *IEEE INFOCOM'96*, San Francisco, March 1996.

[9] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[10] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, CA, April 1994.

[11] P. Goyal, H.M. Vin, and H. Chen. Start-time Fair Queuing: A scheduling algorithm for integrated services. In *Proceedings of the ACM-SIGCOMM 96*, pages 157–168, Palo Alto, CA, August 1996.

[12] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of ACM SIGCOMM'91*, pages 3–15, Zurich, Switzerland, September 1991.

[13] A. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks.* PhD dissertation, Massachusetts Institute of Technology, February 1992.

[14] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. *ACM/IEEE Transactions on Networking*, 1(3):344–357, June 1993.

[15] J. L. Rexford, A. G. Greenberg, and F. G. Bonomi. Hardware-efficient fair queueing architectures for high-speed networks. In *IEEE INFOCOM'96*, San Francisco, March 1996.

[16] S. Suri and G. Varghese and G. Chandranmenon. Leap Forward Virtual Clock. In *Proceedings of INFOCOM 97*, Kobe, Japan, April 1997.

[17] H. Sariowan, R.L. Cruz, and G.C. Polyzos. Scheduling for quality of service guarantees via service curves. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN) 1995*, pages 512–520, September 1995.

[18] S. Shenker. Making greed work in networks: A game theoretical analysis of switch service disciplines. In *Proceedings of ACM SIGCOMM'94*, pages 47–57, London, UK, August 1994.

[19] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of SIGCOMM'95*, pages 231–243, Boston, MA, September 1995.

[20] D. Stiliadis and A. Varma. Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. In *Proceedings of ACM SIGMETRICS'96*, May 1996.

[21] D. Stiliadis and A. Varma. A general mothodology for designing efficient traffic scheduling and shaping algorithms. In *Proceedings of IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[22] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton. A proportional share resource allocation for real-time, time-shared systems. In *Proceedings of the IEEE RTSS 96*, pages 288 – 289, December 1996.

[23] I. Stoica, H. Zhang, and T.S.E. Ng. A Hierarchical Fair Service Curve algorithm for link-sharing, real-time and priority services. In *Proceedings of the ACM-SIGCOMM 97*, Cannes, France, August 1997.

[24] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated Volume 2: The Implmentation*, chapter 24, pages 807–812. Addison-Wesley, 1995.

[25] G.G. Xie and S.S. Lam. An efficient channel scheduler for real-time traffic. Technical Report TR-95-29, University of Texas at Austin, July 1995.

[26] H. Zhang and D. Ferrari. Rate-controlled service disciplines. *Journal of High Speed Networks*, 3(4):389–412, 1994.

A Proof of Theorem 1

Proof.

The approximated ATM algorithm differs from the original WF^{2}Q+in only one case: when a previously idle session i becomes backlogged, its virtual start time should be computed as $\max(F_i(t-), V(t-))$ in WF^{2}Q+, while it will be computed as $\max(F_i(t-), V(t-), (\widehat{S}_{G_i^{tail}} \leq \widehat{S}_g(t) + \frac{L}{r_i}))$ in the approximated system, where $\widehat{S}_g(t)$ and $S_{G_i^{tail}}$ are the virtual start times of the first and last sessions in the group that session i belongs to, respectively. In the following, we will show that the increase of the worst-case delay bound due to this approximation is not more than $\frac{L}{r_i}$.

We call the approximated system $\mathcal{S}$. Consider the system $\mathcal{S}$ with an arbitrary arrival pattern. The idea of the proof is to show, by construction, that there exists a WF^{2}Q+system $\mathcal{S}'$ such that (1) the departure times of all packets are identical in both systems, (2) the time t when a session i becomes backlogged in system $\mathcal{S}'$ is such that $\widehat{S}_i(t) = S_i(t)$. Then, it is easy to see that $\widehat{S}_i(t) = S_i(t)$, $\widehat{F}_i(t) = F_i(t)$, $\widehat{V}_i(t) = V_i(t)$ hold for any time t and session i. In addition, (3) the discrepancy between the arrival times of any packet of session i is bounded by $\frac{L}{r_i}$. The theorem follows directly from (1) and (3).

Let t be the time when a session i becomes backlogged. Let u be the time when the first packet of a backlogged period of session i arrives at the head of the queue. Let $u_1, u_2, \ldots u_N$ be all such events in S, where $u_i < u_{i+1}$, $0 < i < N$. Then we construct a series of systems $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_N$, such that $\mathcal{S}_N = \mathcal{S}'$.

Next, we describe how $\mathcal{S}_1$ is constructed. Let t_1 be the time when the packet that is at the head of the queue at u_1 arrives. Denote this packet p_1, and assume that it belongs to session i. Then t_1 represents the time when session i becomes backlogged in system $\mathcal{S}$. Its start time in $\mathcal{S}$ satisfies the following condition,

$$V(t_1) \leq \widehat{S}_i(t_1) < V(t_1) + \frac{L}{r_i}. \tag{15}$$

Then we define $\mathcal{S}_1$ to be the same as $\mathcal{S}$ except for the arrival time of p_1. More precisely, we choose the arrival time of p_1 to be t_1', such that $V(t_1') = \widehat{S}_i(t_1)$. Since by definition $V(x) - V(y) \geq x - y, \forall x > y$ holds, it is easy to see that t_1' exists and furthermore, $t_1 \leq t_1' < t_1 + \frac{L}{r_i}$. That is, the discrepancy between arrival times of p_1 in the two systems is bounded by $\frac{L}{r_i}$.

The key observation is that replacing t_1 by t_1' does not change u_1. Thus, the virtual time in both $\mathcal{S}$ and $\mathcal{S}'$ remains identical. Further, it is easy to see that the start and finish times of every packet also remain

unchanged, and as a result the packets' departure times are identical in both systems. By applying the same transformation repeatedly we end up obtaining system $\mathcal{S}'$.

B Proof of Theorem 2

Let $B(t)$ be the set of sessions backlogged in the packet system at time t. Let $I(t)$ be the set of sessions that are idle in the packet system at time t. Let A be the set of all sessions at the link. (i.e. $A = B(t) \bigcup I(t)$) In order for the sessions to be granted a rate guarantee, it is necessary that the server is not overbooked. Formally:

$$\sum_{i \in A} r_i \leq C, \tag{16}$$

where r_i is the rate of session i and C is the link capacity.
While the system virtual time function is used to reset the virtual start times of formerly idle sessions, we redefine the assignment of the session virtual start times so as to be properly defined when they are idle:

$$S_i(t) = \begin{cases} \max(V(t), S_i(t-)) & i \in I(t) \\ S_i(t-) + \frac{L_i^k}{r_i} & p_i^k \text{ finishes service} \end{cases} \tag{17}$$

$$F_i(t) = S_i(t) + \frac{L_i^k}{r_i} \tag{18}$$

Recall, our packet system computes a set of approximate virtual start $\widehat{S}_i(t)$ and virtual finish $\widehat{F}_i(t)$ timestamps from the session's computed virtual start $S_i(t)$ and virtual finish $F_i(t)$ timestamps on a per-packet basis.

While subsequent computed timestamps are computed as per Eqs. (17) and (18), the system virtual time is computed based on the approximated timestamps, i.e.

$$V_{WF^2Q+}(t+\tau) = \max(V_{WF^2Q+}(t) + \tau, \min_{i \in \widehat{B}(t+\tau)} (\widehat{S}_i(t+\tau))) \tag{19}$$

where $\widehat{B}(t)$ is the set of sessions backlogged in the WF^{2}Q+ system at time t.

Lemma 1 *Consider an arbitrary interval $[t_1, t_2]$ during a server busy period, and a subset of sessions $H \subseteq A$ which includes all sessions that receive service during $[t_1, t_2]$. If*

$$\sum_{i \in H} S_i(t_1) r_i \geq V(t_1) \sum_{i \in H} r_i, \tag{20}$$

then we have

$$\sum_{i \in H} S_i(t) r_i \geq V(t) \sum_{i \in H} r_i \quad \forall t \in [t_1, t_2] \tag{21}$$

Proof. The proof is by induction on the events in the system.

Base Step. See Ineq. (20).

Induction Step. We consider three cases: (1) a packet departs, (2) a session becomes backlogged, and (3) a session becomes idle. These are the only instances when the system virtual time or the session's start time may change.

Case 1. A packet departs. Assume the kth packet of session j completes service at time $t + \tau$. Then we need to show that

$$\sum_{i \in H} S_i(t+\tau) r_i \geq V(t+\tau) \sum_{i \in H} r_i. \tag{22}$$

From Eq. (19) we have

$$V(t+\tau) \sum_{i \in H} r_i = \max \left(V(t) + \frac{L_j^k}{C}, \min_{i \in \widehat{B}(t+\tau)} (\widehat{S}_i(t+\tau)) \right) \sum_{i \in H} r_i, \tag{23}$$

We consider two subcases, based on the result of the max() operator. If the second element is the largest, Ineq. (22) holds trivially, since by the definition of $V(t+\tau)$, $\widehat{S}_i(t+\tau) \geq V(t+\tau)$ for any backlogged session $i \in \widehat{B}(t+\tau)$ and by Eq. (8) it follows that $S_i(t+\tau) \geq V(t+\tau)$, and since by Eq. (17) we also have $S_i(t+\tau) \geq V(t+\tau)$ for any idle session. Thus, $S_i(t+\tau) \geq V(t+\tau)$, $\forall i \in A$, and consequently $S_i(t+\tau) \geq V(t+\tau)$, $\forall i \in H$.

If the first element is the largest then

$$V(t+\tau) \sum_{i \in H} r_i = V(t) \sum_{i \in H} r_i + \frac{L_j^k}{C} \sum_{i \in H} r_i \geq V(t) \sum_{i \in H} r_i + L_j^k \tag{24}$$

Finally by Eq. (17) we have

$$\sum_{i \in H} S_i(t+\tau) r_i = \sum_{i \in H \setminus \{j\}} S_i(t) r_i + \left(S_j(t) + \frac{L_j^k}{r_j} \right) r_j = \sum_{i \in H} S_i(t) r_i + L_j^k. \tag{25}$$

The proof of this case follows from induction hypothesis, (24), and (25).

Case 2. A session becomes backlogged. Since the start time of a session does not change as the session becomes backlogged (see Eq. (17)), this case follows immediately.

Case 3. A session becomes idle. According to Eq. (17), the start time of a session can only increase (to $V(t)$) when the session becomes idle. As a result, the left hand side of Ineq. (21) increases as the right hand side remains unchanged. This completes the proof of the Lemma.

Let $W_i(t_1, t_2)$ be the amount of session i traffic served in the interval $[t_1, t_2]$

Lemma 2 *Let t and t_1, where $t < t_1$, be two arbitrary time instances when a packet departs. The service received by session i during the time interval $[t, t_1]$ is then bounded as follows*

$$W_i(t, t_1) \leq r_i(S_i(t_1) - S_i(t)) \tag{26}$$

Proof. Let L_i^1, L_i^2, ... L_i^M denote the lengths of the packets of session i transmitted during $[t, t_1]$, if any. By the definition of W_i we have

$$W_i(t, t_1) = \sum_{k=1}^{M} L_i^k \tag{27}$$

Let S_i^k and F_i^k be the start, respectively, the finish time of the k-th packet of session i that is served during the time interval $[t, t_1]$. Then by the definition of the start and finish times (see Eqs. (17) and (18)) it is easy to see that

$$S_i(t) \leq S_i^1 < F_i^1 \leq S_i^2 < F_i^2 \leq \ldots \leq S_i^M < F_i^M \leq S_i(t_1). \tag{28}$$

In addition from Eq. (18) we have

$$L_i^k = (F_i^k - S_i^k) r_i. \tag{29}$$

By combining Eqs. (27), (28), and (29) we obtain

$$W_i(t, t_1) = \sum_{k=1}^{M} L_i^k = \sum_{k=1}^{M} (F_i^k - S_i^k) r_i \leq r_i(S_i(t_1) - S_i(t)). \tag{30}$$

Lemma 3 *Let t_D be the time when the k-th packet of session j departs. Let t_B be the last time before t_D when the following conditions hold:*

1. *a packet departs*
2. *during the entire interval $[t_B, t_D]$ session j has an eligible packet*

The service time received by session j during $[t_B, t_D]$ is then bounded as follows

$$W_j(t_B, t_D) \geq r_j(t_D - t_B - \beta_j^{max} - \gamma_j^{max}), \tag{31}$$

Proof. Note that if packet $k-1$ departs before its finish time, t_B represents the time when packet k becomes eligible and the server can send a new packet.

By the definition of t_B we have

$$\widehat{S}_j(t_B) \quad \leq \quad V(t_B) \tag{32}$$

Let $\widehat{F}_j^k$ be the finish time of the k-th packet of session j in the virtual timespace. Since session j is assumed to have always an eligible packet during the interval $[t_B, t_D]$ the server is continuously busy during the same interval. Thus, during the interval $[t_B, t_D]$, the server allocates

$$W(t_B, t_D) = C(t_D - t_B). \tag{33}$$

If session i is idle at time t_D, let t_D^i denote the time when it became idle, where obviously $t_D^i < t_D$. For uniformity, if session i is backlogged at time t_D, we take $t_D^i = t_D$. Since session i is assumed to be idle during $[t_D^i, t_D]$, we have $W_i(t_B, t_D) = W_i(t_B, t_D^i)$. Further, by Lemma 2 we have $W_i(t_B, t_D) \leq r_i(S_i(t_D^i) - S_i(t_B)$. From here and from Eq. (33), we obtain

$$\begin{aligned} W(t_B, t_D) &= \sum_{i \in A} W_i(t_B, t_D) = \sum_{i \in A \setminus \{j\}} W_i(t_B, t_D) + W_j(t_B, t_D) \Rightarrow \\ C(t_D - t_B) &\leq \sum_{i \in A \setminus \{j\}} r_i(S_i(t_D^i) - S_i(t_B)) + W_j(t_B, t_D). \end{aligned} \tag{34}$$

Rearranging terms we obtain

$$W_j(t_B, t_D) \geq C(t_D - t_B) - \sum_{i \in A \setminus \{j\}} r_i(S_i(t_D^i) - S_i(t_B)) \tag{35}$$

Let H be the set of all sessions that receive service during the interval $[t_B, t_D]$. Then we have

$$\begin{aligned} \sum_{i \in A} r_i(S_i(t_D^i) - S_i(t_B)) &= \sum_{i \in H} r_i(S_i(t_D^i) - S_i(t_B)) \Rightarrow \\ \sum_{i \in A \setminus \{j\}} r_i(S_i(t_D^i) - S_i(t_B)) &= \sum_{i \in H \setminus \{j\}} r_i(S_i(t_D^i) - S_i(t_B)). \end{aligned} \tag{36}$$

Next note that $S_i(t_D^i)$ represents the finish time of the last packet of session i served during $[t_B, t_D]$. Since WF^{2}Q+ selects the packet with the smallest approximate finish time, it can be shown that $S_i(t_D^i) \leq \widehat{F}_j^k$, $\forall i \in H \setminus \{j\}$. Assume this is not true, i.e., there is a session $i \in H \setminus \{j\}$ such that $S_i(t_D^i) > \widehat{F}_j^k$. But this means that the packet with the finish time $S_i(t_D^i)$ was transmitted during $[t_B, t_D]$. Since the approximate finish time of this packet is no smaller than $S_i(t_D^i)$, and since session j is assumed to have an eligible packet at any time during the interval $[t_B, t_D]$ (all of them having a finish time no larger than $\widehat{F}_j^k < S_i(t_D^i)$), this

contradicts the WF^{2}Q+ selection policy and therefore proves our claim, i.e., $S_i(t_D^i) \leq \widehat{F}_j^k$, $\forall j \in H$. With this, Eq. (36) becomes

$$\sum_{i \in H \setminus \{j\}} r_i (S_i(t_D^i) - S_i(t_B)) \leq \sum_{i \in H \setminus \{j\}} r_i (\widehat{F}_j^k - S_i(t_B)), \tag{37}$$

By combining Ineqs. (35) and (37) we obtain

$$W_j(t_B, t_D) \geq C(t_D - t_B) - \sum_{i \in H \setminus \{j\}} r_i (\widehat{F}_j^k - S_i(t_B)) \tag{38}$$

Let $t' < t_B$ be the last time when a packet belonging to a session in $G = A \setminus H$ is served before t_B. Note that any session $i \in H$ that is backlogged at time t' has an approximate finish time no larger than $\widehat{F}_j^k$ (otherwise it would belong to H). Since no such session is served at time t' it follows that all sessions in H are either idle, or the packets at the head of their queues are not eligible at t'. By the definition of the approximate start time and of the eligibility criterion we then have

$$S_i(t') \geq \widehat{S}_i(t') \geq V(t'), \quad \forall i \in H, \tag{39}$$

and further

$$\sum_{i \in H} S_i(t') r_i \geq V(t') \sum_{i \in H} r_i. \tag{40}$$

By applying Lemma 21 we obtain

$$\sum_{i \in H} S_i(t_B) r_i \geq V(t_B) \sum_{i \in H} r_i. \tag{41}$$

By combining Ineqs. (32) and (41) and using the fact that $\widehat{S}_j(t) \geq S_j(t) - \beta_j^{max}$, we obtain

$$\sum_{i \in H \setminus \{j\}} S_i(t_B) r_i + \beta_j^{max} r_j \geq V(t_B) \sum_{i \in H \setminus \{j\}} r_i. \tag{42}$$

From Ineqs. (38) and (42) we get

$$W_j(t_B, t_D) \geq C(t_D - t_B) - \sum_{i \in H \setminus \{j\}} r_i (\widehat{F}_j^k - V(t_B)) - \beta_j^{max} r_j. \tag{43}$$

Further, by Eqs. (17) and (14) we have $S_j(t_D) = F_j^k > \widehat{F}_j^k - \gamma_j^{max}$. Since session j is continuously backlogged during $[t_B, t_D]$ it follows that

$$W_j(t_B, t_D) = r_j (S_j(t_D) - S_j(t_B)) > r_j (\widehat{F}_j^k - \gamma_j^{max} - S_j(t_B)). \tag{44}$$

By combining with Eq. (32) and using the fact that $S_j(t_B) - \beta_j^{max} < \widehat{S}_j(t_B)$ we obtain

$$\widehat{F}_j^k - V(t_B) < \frac{W_j(t_B, t_D)}{r_j} + \gamma_j^{max} + \beta_j^{max}. \tag{45}$$

Finally, by combining with Eq. (43) and using the fact that $\sum_{i \in H} r_i \leq C$ we have

$$\begin{aligned} W_j(t_B, t_D) &\geq C(t_D - t_B) - \frac{C - r_j}{r_j} W_j(t_B, t_D) - (C - r_j)(\beta_j^{max} + \gamma_j^{max}) - r_j \beta_j^{max} \Rightarrow \\ W_j(t_B, t_D) &> r_j(t_D - t_B - \beta_j^{max} - \gamma_j^{max}) \end{aligned} \tag{46}$$

Proof. of Theorem 2

Consider the moments of time t_B and t_D as defined in Lemma 3, where t_D is the departure time of the k-th packet of flow j. By the definition of t_B session j has no eligible packet at time $t_B - \tau$, where τ represents the service time for the packet that departs at time t_B. This means the the packet at the head of the queue of session j becomes eligible at a time $t' \in [t_B - \tau, t_B]$.

Let $L_j^1,\ L_j^2, \ldots, L_j^k$ be the lengths of the packets of session j which are served during $[t_B, t_D]$. From Lemma 3 we have

$$W_j(t_B, t_D) = \sum_{m=1}^{k} L_j^m \geq r_j(t_D - t_B - \beta_j^{max} - \gamma_j^{max}). \tag{47}$$

On the other hand, since session j is continuously backlogged during the interval $[t_B, t_D]$ the deadline of the k-th packet, denoted d_k, is

$$d_k = t' + \frac{\sum_{m=1}^{k} L_j^m}{r_j} > t_B - \tau + \frac{\sum_{m=1}^{k} L_j^m}{r_j}. \tag{48}$$

By combining Ineqs. (47) and (48) we have $t_D < d_k + \tau + \beta_j^{max} + \gamma_j^{max}$

Q.E.D.