A MINIMUM-WORK WEIGHTED FAIR QUEUING ALGORITHM FOR GUARANTEED END-TO-END QUALITY-OF-SERVICE IN PACKET NETWORKS

by

HAITHAM FAHMI TAYYAR

B.Sc. (Electrical Engineering and Mathematics), KFUPM, 1993
M.Sc. (Electrical Engineering), KFUPM, 1997

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

Department of Electrical and Computer Engineering

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October 2002

Department of ELECTRICAL AND COMPUTER ENGINEERING

The University of British Columbia
Vancouver, Canada

Date Oct 9th, 2002

ABSTRACT

Emerging applications in multimedia communications and Virtual Private Networks (VPNs) require data networks to provide Quality-of-Service (QoS) guarantees, such as delay and/or jitter bounds, to individual packet flows. Providing such guarantees can be achieved by link scheduling mechanisms along the path of these packets.

Among the many packet-scheduling techniques proposed for this problem, Weighted Fair Queuing (WFQ) offers the best delay and fairness guarantees. However, all previous work on WFQ has been focused on developing inefficient approximations of the scheduler because of perceived scalability problems in the WFQ computation.

This thesis proves that the previously well accepted O(N) time-complexity for WFQ implementation, where N is the number of active flows handled by the scheduler, is not true. The other key contribution of the thesis is a novel Minimum-Work Weighted Fair Queuing (MW-WFQ) algorithm which is an O(1) algorithm for implementing WFQ. In addition, the thesis presents several performance studies demonstrating the power of the proposed algorithm in providing precise delay bounds to a large number of sessions with diverse QoS requirements, whereas other well known scheduling techniques have failed to provide the same guarantees for the same set of sessions.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

My sincere thanks to The University of British Columbia and The Department of Electrical and Computer Engineering for providing me with a friendly atmosphere of research and knowledge. Also, I would like to acknowledge NSERC, ASI and all government agencies that supported my research.

I would like to extend a warm thank you to my advisor and close friend Dr. Hussein Alnuweiri. His endless patience, generous support and insight has made this thesis possible. I would also like to thank Dr. Victor Leung for his continued support and funding of my research.

My love and appreciation goes out to my parents, my brothers and my sister, whom were a constant source of motivation and support. Their love and care carried me through some difficult moments in my life. Their prayers, guidance and inspiration lead to this accomplishment.

I would like to express my profound gratitude and appreciation to my thesis committee for their valuable time, constructive criticism, and stimulating discussions.

The many friends I have made throughout my stay here in Vancouver made it a wonderful experience. I especially would like to thank Zoheir, Ahmed, Wesam (aka W), Tamer, Ayman, Amr, Awad and Anwar for being such good friends and role models. I would also like to thank my gym partners and friends Dan and Kamal for putting up with me for all these years.

My thanks to the city of Vancouver and the people of Canada for their hospitality

and for giving me the opportunity to call this place home.

I extend my love and deep appreciation to the person who has stood beside me and made this dream a reality, my wife Iman. I thank her for her patience and for giving me a lovely family. I also thank her parents and sisters for embracing me as one of their own.

Last but not least, all praise and thanks is due to almighty god, for his limitless blessing and guidance.

Dedicated with Love

to

my parents Fahmi and Farida,

my wife Iman, and my son Yamin.

Chapter 1

Introduction and Motivation

1.1 Introduction

Applications with strict Quality-of-Service (QoS) guarantees, such as bounded delay and/or jitter, require the enforcement of some form of scheduling discipline along the path of packets generated by these applications. Scheduling methods play a key role in providing QoS guarantees in packet networks. They allow packets with strict delay deadlines to move ahead of packets with looser deadline requirements, independent of when these packets arrived. Scheduling methods vary in methodology and assumptions. In general, there are two classes of scheduling methods: (a) per-flow scheduling, and (b) aggregate scheduling.

In per-flow scheduling, a stream is assumed to have its own separate queue which contains only its packets. When the link becomes idle and there are packets in the

queues, the scheduler arbitrates between the different stream queues choosing the packet that is to leave the link next (depending upon delay deadlines). In aggregate scheduling on the other hand, streams are aggregated and per-flow-scheduling is applied to these aggregates.

Both per-flow and aggregate scheduling methods deal with "streams" or "flows". A flow can be viewed as a sequence of packets having a set of common characteristics. These characteristics vary depending on the way flows are classified and where in the network they are being classified. For example, a common way to classify flows is a combination of the source and destination IP address, the source and destination port number and possibly the application generating the packets. The reason behind such classification is that, traffic generated by a particular application from a given host and destined to another host usually has similar QoS requirements. For example, packets of a Telnet session from one host to another require the same kind of treatment and can be considered a single session. Figures 1.1 and 1.2 show the basic differences between per-flow and aggregate scheduling.

A potentially large number of flows can be active during any period of time in the Internet. It is assumed that the core part of a large network, such as the Internet, cannot handle such a large number of flows. Therefore, individual flows may need to be merged into larger ones inside the core. This scalability issue led people to investigate new techniques capable of providing flows in the Internet with a service that is better than the current "Best-effort" but does not require maintenance of

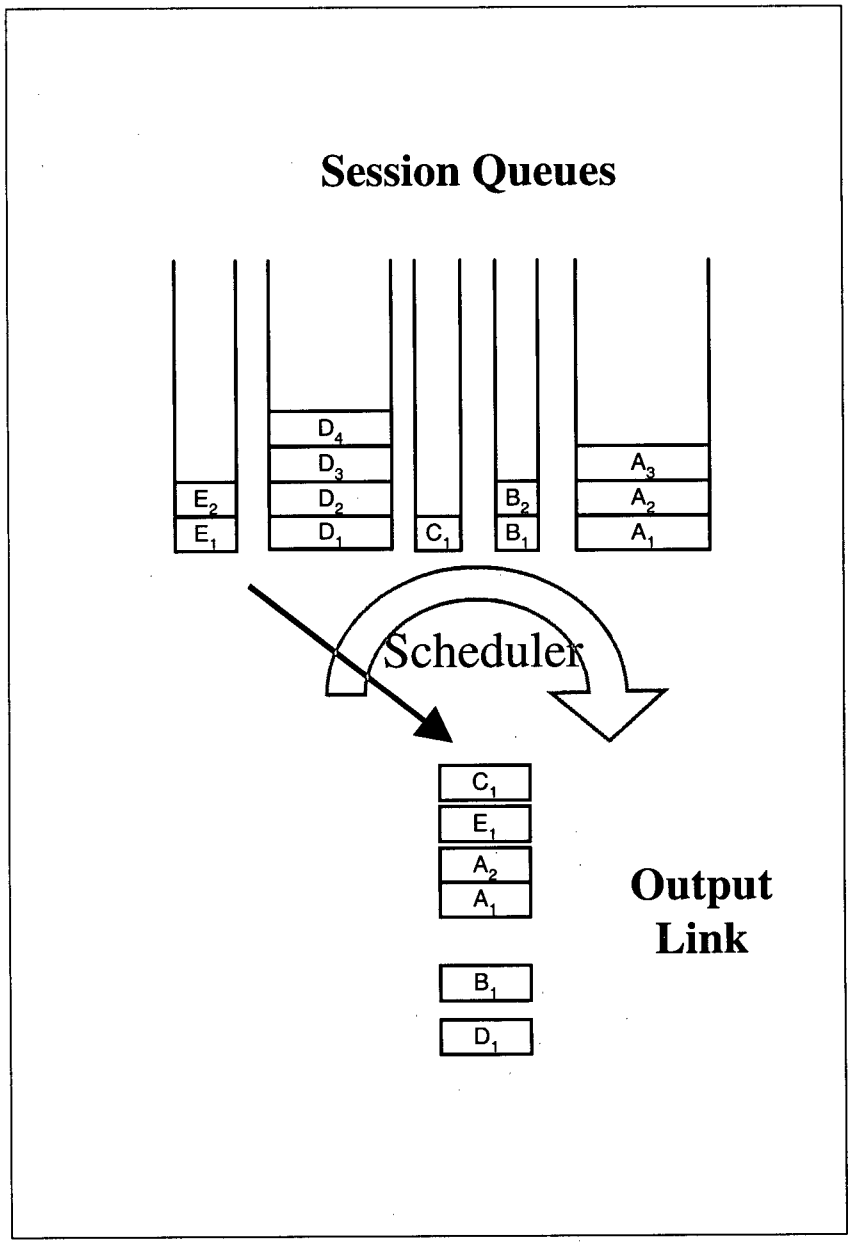

Figure 1.1: Per-flow scheduling

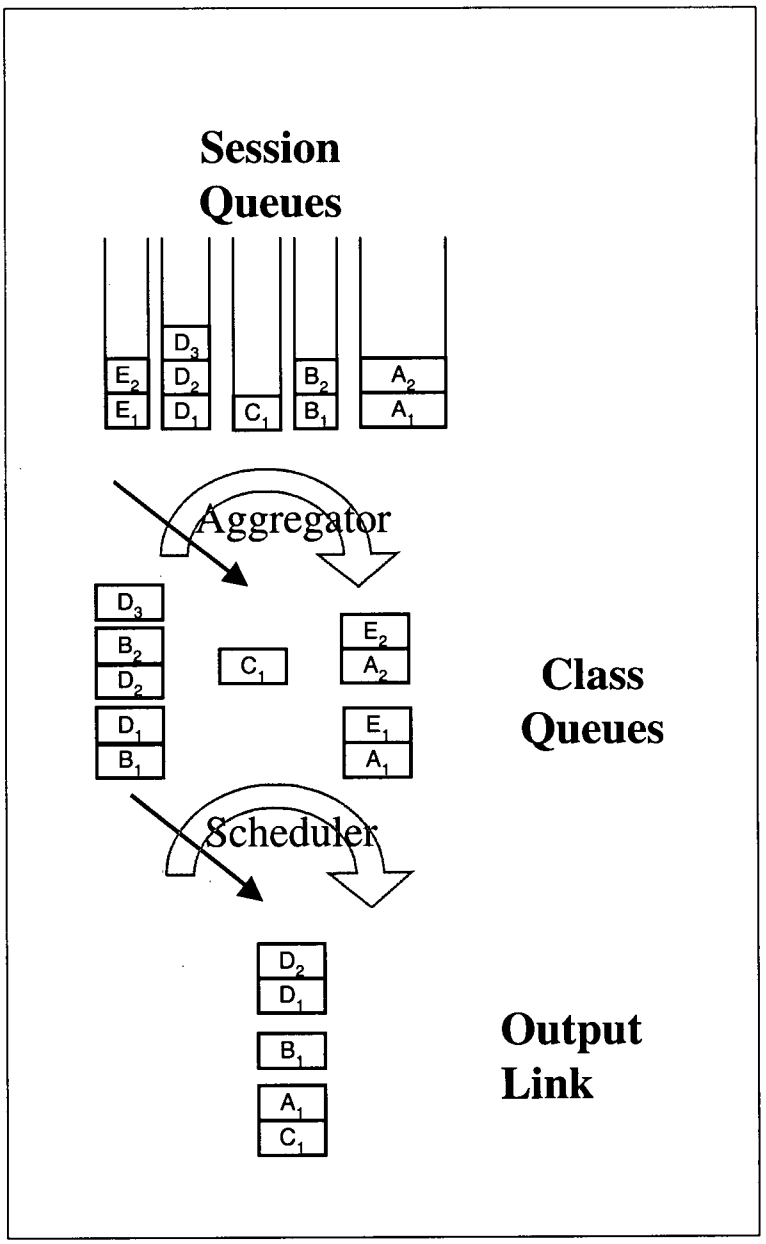

Figure 1.2: Aggregate scheduling

per-flow information. This was the motivation for the Differentiated Services or Diffserv framework [1]. In this framework, traffic is classified according to its per-hop behavior and placed into a finite set of priority classes. Traffic in a class is serviced using FIFO scheduling. If there are packets in the higher priority classes, they are serviced first before packets of lower priority classes. In Diffserv, the highest priority traffic is aggregated into a class called the Expedited Forwarding (EF) class.

People have proposed many different kinds of per-flow and aggregate scheduling techniques to handle the problem of providing end-to-end guarantees for traffic with strict QoS requirements. Although some delay bounds have been defined for different aggregate scheduling methods, they are either weak or based on some strict shaping assumptions that may not be enforceable in large-scale networks [2]. On the other hand, per-flow techniques provide more strict guarantees but are challenged by the scalability problem in managing a large number of flows in the network.

The importance of per-flow scheduling lies in the fact that these techniques provide strict end-to-end delay and fairness guarantees to individual flows or sessions. In the framework of Multiprotocol Label Switching (MPLS), this provides a sure way of choosing the end-to-end paths during Label Switched Path (LSP) setup [3]. Currently, most implementations of MPLS use Diffserv to aggregate flows into LSPs which are then mapped to a certain per-hop behavior. The nodes in the MPLS domain are then expected to schedule these aggregates based purely on the Diffserv classification of these aggregates. However, if the nodes along the path do not im-

plement per-flow scheduling, then the LSPs will not receive strict end-to-end delay guarantees.

From this point on, when discussing per-flow techniques, a flow may be either an individual session or an aggregate.

1.2 Main Contributions

In this Thesis, the algorithm proposed enables Weighted Fair Queuing (WFQ), one of the most important per-flow schedulers, to scale to large systems where link speeds are high and a potentially large number of sessions is active at any given time. Previous approaches for implementing WFQ have $O(N)$ time complexity, where N is the number of sessions sharing a link, because of which these techniques cannot be implemented in practice. This led people to consider alternatives to WFQ which have reduced QoS guarantees but are less complex to implement. In this thesis, five main contributions are presented; the first is a clear identification of the three main potential causes of $O(N)$ time complexity in standard WFQ implementation. The second is a theoretical analysis that shows that one of these causes is due to an event which cannot be avoided even under the most relaxed conditions. The third is an algorithm which enables the calculation of the virtual time needed to implement standard WFQ. The fourth and the most fundamental contribution of this thesis is an algorithm called Minimum-Work Weighted Fair Queuing (MW-WFQ)

that is based on the standard WFQ implementation but has $O(1)$ time complexity and hence enables the implementation of WFQ. The fifth contribution is extensive experimental testing of the proposed MW-WFQ algorithm that shows clearly that it is able to provide strict end-to-end delay guarantees in packet networks at a low implementation cost.

The work presented in this thesis has a strong impact on the state-of-the-art design of high-speed networking routers and switches. The results of this work show clearly that the reluctance towards using per-flow scheduling in high-speed networks is no longer justifiable on the basis of implementation cost. This thesis forces the networking community to reconsider per-flow scheduling as a viable and efficient solution to the end-to-end QoS problem in the Internet. It is now possible to use MW-WFQ to provide the thousands of LSP's of an MPLS network with precise QoS guarantees that can never be met using simple aggregation techniques like the ones used in Diffserv.

1.3 Thesis Organization

Chapter 2 presents a survey of the most common per-flow scheduling techniques. In particular, Weighted Fair Queuing (WFQ), having the best properties among all the other per-flow methods, will be shown to have $O(N)$ computational complexity, where N is the number of sessions sharing the link. In Chapter 3, it is proven that

this high computational complexity cannot be avoided even under the most relaxed conditions. Chapter 4 covers the proposed algorithm for overcoming the problem of $O(N)$ computational complexity. In Chapter 5, experimental results based on the implementation of the proposed algorithm on a *Linux* router are presented. And finally, Chapter 6 presents the conclusion and opens directions for future work.

Chapter 2

Introduction to Per-flow Scheduling

2.1 Introduction

Packet scheduling as a tool for providing per-flow or per traffic-class Quality-of-Service (QoS) guarantees in packet networks is well-understood and strongly supported by both fundamental theoretical arguments [4, 5], as well as practical tests. The current Internet is based on a best-effort service model that does not provide any QoS assurances to different applications. This lack of service differentiation has serious impact on the type of applications that require end-to-end QoS assurance over the Internet. For example, real-time communications and/or interactive applications over the Internet require resource reservation and scheduling at involved

hosts and intermediate nodes. For such applications, the networks must provide guaranteed rates, bounded end-to-end delays, restricted packet loss, fairness, etc., to individual flows.

With the proper dimensioning of network resources, the most important performance attributes of a packet-scheduling algorithm become its delay and fairness bounds for each flow. Delay bounds are important for a wide range of time-sensitive or real-time services. Fairness bounds are important for providing a sufficient degree of isolation to a flow of packets, so that the service guaranteed to that flow is not affected by the behavior or misbehavior of other packet flows, sharing the same link. To provide such guarantees, it is normally assumed that packet flows have been conditioned using an appropriate traffic shaper, such as a leaky-bucket conditioner, and that the policing is in effect at the network edges.

Providing end-to-end delay bounds to individual flows in a packet network, such as the Internet, requires the use of schedulers that can guarantee packet service rates as well as fair allocation of excess bandwidth. Generalized Processor Sharing (GPS) is an ideal scheduler that provides every flow its guaranteed bit-rate and distributes excess bandwidth fairly among flows according to their relative bandwidth weights. As a result, GPS can provide end-to-end delays and fairness guarantees to packet flows that are shaped by leaky bucket traffic conditioners. GPS works by assigning a distinct queue to each flow (or session), then servicing an infinitesimally small amount from each session according to a weighted cyclical schedule. Unfortunately,

GPS is un-realizable in practice because it services a small part of a packet at a time. A real scheduler must complete the service of an entire packet from a session before it moves to the next session. Packet-by-packet GPS, commonly known as Weighted Fair Queuing (WFQ), is one of the GPS emulation algorithms that transmits packets according to their finish order under GPS [6, 7]. WFQ simulates a GPS fluid-model in parallel with the actual packet-based scheduler in order to calculate the virtual finish number (used as a timestamp) for packets arriving to the scheduler. To calculate the finish number, WFQ maintains the state of the system by means of a *Virtual Time* function $V(t)$ which is a piecewise linear function of real time t, and whose slope changes depending on the number of backlogged sessions and their service rates. To perform scheduling in real-time, WFQ must update the virtual time before any packet arrival, so that every arriving packet gets the proper virtual finish number (as if it will be departing under GPS). The virtual-time function is impacted by arrivals (to empty queues), as well as departures of packets (that result in empty session queues). The problem is that, an undetermined (and possibly large) number of session queues can become empty at the same time, because under GPS many packets can end up having the same virtual finish time. Therefore, updating the virtual time function in between two consecutive packet arrivals may incur a large number of computations. In particular, if a link is shared by up to N active sessions, then updating the virtual time can incur a computation on $O(N)$ sessions or queues. This problem is usually referred to as iterated deletion [6], and is the main reason

why WFQ has not been implemented in practice. The number of active sessions on a Gigabit link can reach several tens or even hundreds of thousands, which translates into a proportional number of computations per packet arrival. Because of the high complexity associated with simulating the GPS system, WFQ has attracted a lot of attention over the past decade [8, 9, 10, 7, 11, 12, 13, 14, 15, 16, 6], and many techniques have been proposed to simplify the virtual time calculations. Some of the key proposals are reviewed in this chapter. In general, such simplifying approaches suffer from either a decrease in fairness (or flow isolation) or an increase in the delay bound.

2.2 Generalized Processor Sharing (GPS)

In GPS, it is assumed that the traffic satisfies a fluid model that assumes that every packet is infinitely divisible. Suppose that there are N sessions sharing an outgoing link of capacity C. The share of bandwidth reserved by session i is represented by a real number α_i. The α's are chosen such that the fraction

$$\frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j} \tag{2.1}$$

corresponds to the desired bandwidth reservation of the session. That is, if ρ_i is the desired bandwidth reservation of session i, then

$$\frac{\alpha_i C}{\sum_{j=1}^{N} \alpha_j} \geq \rho_i \tag{2.2}$$

where the quantity

$$r_i = \frac{\alpha_i C}{\sum_{j=1}^{N} \alpha_j}$$

is called the guaranteed rate for session i and is the minimum bandwidth available to session i at any given instance of time. Also,

$$\overline{\alpha}_i = \frac{r_i}{C} = \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

is the normalized share of session i.

Let $B(\tau, t)$ be the set of sessions that are backlogged in the interval (τ, t). Then, under GPS, the service $W_i(\tau, t)$ offered to a session i that belongs in $B(\tau, t)$ is proportional to α_i. That is

$$W_i(\tau, t) = \frac{\alpha_i}{\sum_{j \in B(\tau,t)} \alpha_j} C(t - \tau) \tag{2.3}$$

GPS attains its bandwidth guarantees by servicing an infinitesimal amount from each backlogged session in proportion to each session's reservation [6]. As a result,

GPS provides perfect isolation, ideal fairness and low end-to-end session delays. Perfect isolation in GPS is due to the fact that when a sessions sends traffic beyond its reserved rate, only that session's packets will experience an increase in delay; the packets from other sessions will continue to receive their fair share. The GPS scheduler provides ideal fairness in the sense that any unused bandwidth which belongs to the idle sessions is distributed fairly among the backlogged sessions in proportion to their shares. Also, because a backlogged session is serviced at least at the minimum guaranteed rate, its packets will receive delay guarantees. It has been shown in [5] that if the inputs to a GPS scheduler are shaped by a token-bucket shaper, then these sessions will experience low end-to-end delay bounds which depend only on their reserved (guaranteed) rate. However, because GPS is based on the fluid model, it is un-implementable, since a scheduling technique will have to serve packets as a whole. In the following section, the most important emulation of GPS is described. In general, all these emulations aim at ordering packets for transmission according to their finish or start times in the ideal GPS system.

2.3 Implementation of GPS in a Packet System

It has been seen before that the GPS scheduler is un-implementable since no packet can be partitioned into infinitesimal quantities. As a result, people decided to use the service order of packets in GPS to schedule packets in a packet system. This

leads to two GPS emulation policies, the Smallest Finish-time First (SFF) and the Smallest Start-time First (SSF). In the SFF techniques, packets are serviced in the order in which they finish under GPS. A SSF technique, on the other hand, services packets according to the starting order under GPS. Weighted Fair Queuing (WFQ) is an example of a SFF scheduler while Start-time Fair Queuing (SFQ) is an example of a SSF scheduler.

In GPS, it is possible for more than one packet to finish at the same time even if they arrive at different times. Hence, at a given time there may be as many as N packets leaving the system. As a result, people have assumed that a GPS system is un-implementable since it requires knowledge of at most N events at a given instance of time [6]. As a result, it was argued that the implementation complexity of any exact GPS emulation is $O(N)$. However, it will be shown later that this is not true since the system states can be updated in $O(1)$ although as many as N events can occur simultaneously. This is achieved by creating a priority queue data structure that keeps track of the finish order of packets in the system. Arrivals and departures only effect this data structure. The time it takes to update the data structure, based on arrivals, is dependent on the time it takes to insert into this data structure. Such an insertion operation can be performed in at most $O(\log N)$ operations when the priority queue is implemented as a heap. The effect of departures on the data structure can be updated in $O(1)$ since it involves only a removal from the head of the data structure. In both cases, it is seen that the complexity of implementing

GPS is at most $O(\log N)$ and not $O(N)$. If a data structure with a faster insertion time is found, it will reduce the complexity of GPS emulation even further. One such data structure is the Calendar Queue [17] which has $O(1)$ insertion and deletion time.

2.4 GPS Emulation

In the following subsections, the most common GPS emulation techniques are discussed.

2.4.1 Weighted Fair Queuing (WFQ)

Packet-by-packet GPS, or WFQ as it is known, is one of the GPS emulations that transmits packets according to their finish order under GPS. In WFQ, a GPS fluid-model system is simulated in parallel with the actual packet-based system in order to identify the set of sessions backlogged [1]at each instant of time and their service rates. Based on this information, a timestamp is calculated for each arriving packet, and the packets are inserted into a priority queue based on their timestamp values and transmitted in order of increasing timestamps. The timestamp specifies the finish number of the packet. The finish number of a packet represents the round number at which the arriving packet will depart the system if the scheduler is a bit-by-bit round-robin scheduler. To calculate the finish number, WFQ keeps track of a virtual time function $V(t)$ which is a piecewise linear function of real time t, and whose slope changes depending on the number of busy sessions in GPS and their service rates. More precisely, if $B(t)$ represents the set of backlogged sessions in the GPS scheduler at time t, then the slope of the virtual time function at time

[1]From now on, whenever backlogged or idle sessions are mentioned, it means those in the corresponding GPS scheduler and not in the actual packet scheduler.

t is given by

$$\frac{\sum_{j=1}^{N} \alpha_j}{\sum_{i \in B(t)} \alpha_i} \tag{2.4}$$

The virtual time function keeps track of the "normalized" service provided by the system to all backlogged sessions. At the arrival of a new packet, the virtual time must first be calculated. Then, the timestamp TS_i^k associated with the k^{th} packet of session i that arrives at time t is calculated as

$$TS_i^k = \max(TS_i^{k-1}, V(t)) + \frac{L_i^k}{r_i} \tag{2.5}$$

Where L_i^k is the length of the arrived packet and r_i is the guaranteed link share of session i.

It has been shown that if arrivals from session i are token bucket shaped with a token bucket rate of r_i and a token bucket depth of σ_i, then the maximum queuing delay a packet of session i experiences in a WFQ scheduler is bounded [4, 8] and equals

$$\frac{\sigma_i}{r_i} + \frac{L_i}{r_i} + \frac{L_{max}}{C} \tag{2.6}$$

where L_i is the maximum packet size of session i, and L_{max} the maximum packet size among all the sessions sharing the link.

The advantage of WFQ is that, it is able to provide the same latency bound of GPS, with a maximum discrepancy equal to the transmission time of one maxi-

mum length packet. However, it has been argued, that this technique has a serious limitation due to the computational complexity arising from the simulation of the fluid-model GPS scheduler [6]. In particular, it has been mentioned that if there are a total of N sessions sharing the outgoing link, a maximum of N events may be triggered in the simulation during the transmission time of single packet. Thus, the time for completing a scheduling decision was considered to be $O(N)$ [6]. As a result, when the number of sessions sharing the outgoing link is large, the simulation of GPS was considered to be prohibitively expensive. In contemporary Internet routers, N can reach several tens of thousands of sessions. However, it will be shown later that in practice the implementation of WFQ is not as complex as previously assumed.

To solve the problem of this perceived computational complexity, several variants of WFQ were proposed, all of which attempt to reduce the computational complexity of the timestamps. However, the true cost of computing the timestamp in these algorithms is equivalent to that of WFQ and there are no apparent savings in computational time by performing such approximations.

2.4.2 Self-Clocked Fair Queuing (SCFQ)

In SCFQ, an approximation of the virtual time function $V(t)$ is calculated using the timestamp of the packet currently in service [12]. Thus, if TS_{cur} denotes the timestamp of the packet currently in service, the virtual time $V(t)$ is taken as TS_{cur}.

SCFQ calculates the timestamp of an arriving packet, say the k^{th} packet of session i, as

$$TS_i^k = \max(TS_i^{k-1}, TS_{cur}) + \frac{L_i^k}{r_i} \tag{2.7}$$

This approach reduces the complexity of the algorithm greatly. However, the price paid is a reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of sessions that share the outgoing link.

2.4.3 VirtualClock (VC)

The VirtualClock scheduling algorithm provides the same end-to-end delay bound as WFQ with a simple timestamp computation algorithm. The virtual time function in this algorithm is the real time and hence the timestamp becomes [11],

$$TS_i^k = \max(TS_i^{k-1}, t) + \frac{L_i^k}{r_i} \tag{2.8}$$

The disadvantage of this algorithm is that a backlogged session can be starved for an arbitrary period of time as a result of excess bandwidth it received from the server when other sessions were idle.

2.4.4 Start-time Fair Queuing (SFQ)

Another variant of WFQ is the Start-time Fair Queuing (SFQ) [16]. This technique tries to schedule packets according to start time in GPS. The virtual time is approximated by the virtual start time of the packet currently in service. Packets are scheduled in order of start-time with ties broken by the toss of a coin. The virtual finish time of a packet is calculated as the sum of the virtual start time plus the ratio of length to session share. When a packet arrives to a backlogged session, its virtual start time is set equal to the virtual finish time of the previous packet of that session. Although SFQ is easier to implement than WFQ, it has a delay bound well above that of WFQ.

2.4.5 Rate-Proportional Server techniques (RPS)

To solve these problems, three new variants of WFQ, namely Frame -Based Fair Queuing (FFQ) [10], Starting Potential-Based Fair Queuing (SPFQ) [10] and Minimum-Delay Self-Clocked Fair Queuing (MD-SCFQ) [14] have been proposed. These algorithms have the delay bounds of WFQ, bounded unfairness and simple timestamp computation. These three algorithms can be understood through the theory of Rate-Proportional Servers (RPS) and the concept of Potential Function (which is a generalization of virtual time in GPS.) The RPS can be described in terms of Latency-Rate (LR) Servers [8].

In RPS schedulers, a potential is associated with every session and is incremented in such a way as to provide a fair service to each session based on its associated rate. Assume that N sessions share the outgoing link, each with an associated rate r_i, and that the total bandwidth assigned to the sessions does not exceed the link capacity C or:

$$\sum_{i=1}^{N} r_i \leq C \tag{2.9}$$

When a session i is backlogged, its potential increases exactly by the normalized service it receives. That is, if $P_i(t)$ denotes the potential of session i at time t, then, during any interval $(\tau, t]$ within a backlogged period for session i

$$P_i(t) - P_i(\tau) = \frac{W_i(\tau, t)}{r_i} \tag{2.10}$$

Where $W_i(\tau, t)$ denotes the amount of service received by session i during the interval $(\tau, t]$.

A server is defined to be a Rate-Proportional Server (RPS) if it attempts to equalize the potential of all backlogged sessions at every instance of time. This is achieved in the fluid server as follows: at any instant t, the scheduler services only the subset of sessions with the minimum potential, and each session in this subset receives service in proportion to its reserved rate r_i. In this way the scheduler increases the potentials of the sessions in this subset at the same rate. When a session becomes backlogged, its potential is updated based on a *system potential*

function that keeps track of the progress of the total work done by the scheduler. The system potential $P(t)$ is a nondecreasing function of time. When an idle session i becomes backlogged at time t, its potential $P_i(t)$ is set to:

$$P_i(t) = \max[P_i(t^-), P(t)] \tag{2.11}$$

to account for the service it has missed [8]. The difference between RPS schedulers is in the way they update the system potential. This gives rise to scheduling techniques with varying delay and fairness behaviors.

The general requirements for a function to be a system potential is that it never exceeds the potential of any backlogged session

$$P(t) \leq P_i(t) : \forall i \in B(t) \tag{2.12}$$

Where $B(t)$ denotes the set of sessions that are backlogged in the server at time t. If this requirement is relaxed, then a session with a potential less than the system potential may get exclusive control of the server for a period of time which violates the fairness requirement. In the RPS system, a system potential function must satisfy the additional requirement that during any interval $(t_1, t_2]$ within a system busy period, the system potential function must be increased with a rate of at least

one, that is

$$P(t_2) - P(t_1) \geq (t_2 - t_1) \tag{2.13}$$

An RPS server is a Latency-Rate (LR) Server with zero latency.

Since RPS assumes a fluid model which is not true in packet servers, a version of RPS for packet systems is defined and is called Packet-By-Packet RPS (PRPS) [8]. In these servers the timestamp calculation is as follows: Let us assume that when the k^{th} packet from session i finishes service in the fluid server, the potential of session i is TS_i^k. This finishing potential can be used to timestamp packets and schedule them in increasing order of their timestamps. It can be shown that the service offered by the PRPS to a session can never lag behind that of the fluid RPS by more than one packet. As a result, the latency bound of any PRPS is identical to that of WFQ.

The fundamental difficulty in designing a practical PRPS is the need to maintain the system potential function. In order to avoid simulating the fluid-model RPS in parallel and maintaining its system potential function, the system potential function can be an approximation of this fluid-model system potential and only needs to be updated when a packet departs from the system. One way of doing this is to define a reference potential function $S^P(t)$ called the *base potential*, and calibrate the system potential according to this function at the instances when a packet departs the system [9]. The system potential function is maintained as a piecewise

linear function with a slope of 1 for each linear segment, but calibrated periodically. When the system is not busy the system potential function is equal to zero. During a system busy period, the function is a piecewise linear function of time t. Let τ_0 be the beginning of the current system busy period, then:

1. At times $\tau_1, \tau_2, \ldots, \tau_k$ with $\tau_1 < \tau_2 < \ldots < \tau_k$, a re-calibration is performed by updating $P(t)$ to the base potential $S^P(t)$ at that instant, if the system potential is lower than the base potential. That is, $P(\tau_j) = \max(P(\tau_j^-), S^P(\tau_j))$, where τ_j denotes the instant of time just before the update.

2. At any time t between updates, the system potential increases linearly with time. That is, $P(t) = P(\tau_j) + (t - \tau_j), \tau_j \leq t < \tau_{j+1}$.

The base potential function S^P is a non-decreasing function of time with the following two properties. First, its value at any time is never higher than the potential of any backlogged session at that instant. Second, the difference between the base potential and the potential of any backlogged session is bounded at any time. As a result, this system will be a PRPS and consequently have the same latency bound of WFQ.

Frame-Based Fair Queuing (FFQ)

If the interval between successive re-calibrations is bounded, the scheduler will have bounded unfairness by bounding the difference between the system potential and the

potentials of backlogged sessions. By choosing different base potential functions and re-calibration intervals, we can have different algorithms, all with a latency bound equal to that of WFQ. One such algorithm is FFQ that updates the system potential at regular intervals [10]. It has an upper bound on the period of calibration defined in terms of an internal parameter of the system called the frame size F. The frame size F is defined such that exactly F bits can be transmitted during a frame period T. That is

$$T = \frac{F}{C} \tag{2.14}$$

Φ_i is defined as

$$\Phi_i = \overline{\alpha}_i F = r_i T \tag{2.15}$$

where, $\overline{\alpha}_i$ is the normalized share of session i and Φ_i defines the maximum amount of session i traffic that can be serviced during one frame. When a session remains backlogged, its potential increases by the normalized service offered to it. Thus, when Φ_i bits are serviced from session i, its potential will increase by

$$\frac{\Phi_i}{r_i} = T \tag{2.16}$$

A restriction that the largest packet of a session can be transmitted during a frame period is imposed. That is, if L_i^{max} is the maximum packet size for session i, then

$$L_i^{max} \leq \Phi_i \tag{2.17}$$

In FFQ the base potential function $S^P(t)$ is defined as follows: $S^P(t)$ is a step function whose value is zero when the server is idle and increases by T on every frame update instant. Therefore, at the k^{th} frame update instant τ_k, $S^P(t)$ assumes a value of kT.

Now, defining the starting potential of a packet j of session i as the potential of session i when packet j starts being serviced in the corresponding fluid server, the scheduler can keep track of all the sessions that are backlogged and have packets with starting potential in the next frame. When the starting potentials of the packets at the head of the queues of all backlogged sessions have crossed the frame boundary, the potentials of the sessions in the fluid system have also crossed the frame boundary. Therefore, the crossing time of the last session is a valid time to update the frame and system potential function. On the arrival of a packet, the current system potential is obtained by adding to P the elapsed real time since the current packet in service started transmission. The starting potential of the newly arrived packet is then computed as the maximum of the finishing potential of the previous packet from the same session and the system potential. The packet is then

time stamped with its finishing potential based on its length and the reserved rate. If the starting and finishing potentials of the packet belong to different frames, the current packet is one that crosses over to the next frame. Therefore, the packet is marked to indicate that this is the first packet of the session to cross over to the next frame. The algorithm maintains one counter per frame to keep track of the number of sessions whose packets cross into the next frame. Later, when a marked packet is scheduled for transmission, the corresponding counter is decremented; when the counter reaches zero, the potentials of all the backlogged sessions have crossed over to the next frame, and a frame update can be performed.

When a packet finishes transmission, the system potential is first increased by the transmission time of the packet just serviced. The packet with the minimum timestamp is then selected for transmission. If the transmitted packet was marked, the counter corresponding to the current frame is decremented. If the counter becomes zero, the session that was serviced is the last to cross the current frame. If in addition, the timestamps of none of the queued packets fall in the current frame, a frame update is then performed by incrementing the frame number and re-calibrating the system potential to the corresponding base potential.

Starting Potential-Based Fair Queuing (SPFQ)

Alternatively, the system potential function can be updated every time a packet departs the system. This variant of FFQ is called Starting Potential-Based Fair

Queuing (SPFQ) [10]. In this algorithm, the base potential function $S^P(t)$ is defined as:

$$S^P(t) = \min_{i \in B^P(t)} S_i(t) \tag{2.18}$$

where $B^P(t)$ denotes the set of backlogged sessions in the packet server at time t and $S_i(t)$ the starting potential of the first packet in the queue of a backlogged session i in the packet server. The algorithm executed after packet arrival and departure in SPFQ is similar to that of FFQ except that no counters are used and the base potential function is maintained by keeping track of the minimum starting potential of all head packets of queues of backlogged sessions. Because SPFQ updates the system potential more frequently, it has better fairness properties than that of FFQ.

Mininmum-Delay Self Clocked Fair Queuing (MD-SCFQ)

This algorithm is similar to SPFQ but with the base potential function $S^P(t)$ defined as:

$$S^P(\tau_j) = \frac{\overline{F}_{B(\tau_j)} - L_{B(\tau_j)}}{\tau_{B(\tau_j)}} \tag{2.19}$$

where

$$\overline{F}_{B(\tau_j)} = \sum_{i \in B(t_j)} F_i r_i$$

is the weighted sum of the timestamps of all sessions that are backlogged at time τ_j with each timestamp weighted according to the reserved service rate of the cor-

responding sessions.

$$L_{B(\tau_j)} = \sum_{i \in B(\tau_j)} l_i$$

is the sum of the lengths of the packets at the head of each session queue at time τ_j including the packet that is currently being transmitted. Also,

$$\tau_{B(\tau_j)} = \sum_{i \in B(\tau_j)} r_i$$

is the cumulative service rate of all the sessions that are backlogged at time τ_j where τ_j is the j^{th} re-calibration time of the system potential function [10].

FFQ, SPFQ and MD-SCFQ all have the same delay bound of WFQ under leaky bucket traffic. Their fairness properties can be made close to that of WFQ. The only difference is that the timestamp computation in MD-SCFQ is a lot easier since it does not require any calculation of session potentials and starting-time potentials. However, MD-SCFQ still requires maintaining a sorted priority queue and inserting and deleting an element into this queue every time a packet arrives or departs. In reality, however, the computational simplification in timestamp calculation is not significant when compared with WFQ. The reason is that the quantities $\overline{F}_{B(\tau_j)}$, $L_{B(\tau_j)}$ and $\tau_{B(\tau_j)}$ need to be maintained whenever a packet arrives to an idle session or a packet departs from the system. This requires similar computations as calculating the timestamps in WFQ. Both WFQ and MD-SCFQ need to maintain a system potential function. However, WFQ does not need to maintain individual potentials

as in MD-SCFQ. In WFQ, the system potential function (virtual time) is maintained in a few steps whenever a session becomes newly backlogged or a departure from a session occurs. This means that a packet can get its timestamp quickly. The only time-consuming operation in WFQ is the insertion into the priority queue of timestamps. However, this is required in all sorted-priority schedulers including MD-SCFQ. Thus, both WFQ and MD-SCFQ have comparable implementation costs with WFQ having the advantage of being the most accurate GPS-emulation method. Therefore, if a GPS-based scheduler that provides fairness is to be implemented, then WFQ itself, and not an approximation of it, should be implemented.

2.4.6 Other Techniques based on Potential Functions

Discrete-Rate scheduling [15] is a technique that aggregates all the flows with a given rate or whose rate is a multiple of a given rate into a single flow. Discrete-rate scheduling assumes that the basic rates are finite and hence all the flow rates in the system are either equal to these rates or are multiples of them. The aggregated flows are then serviced using a GPS-emulation technique such as MD-SCFQ. This technique is applicable mainly to ATM systems and has the advantage of reducing the system states. However, the main disadvantage of this technique is the assumption that a set of basic rates exists for all the flows in the system. Also, it should be noted that this technique uses a GPS-approximation and hence results in higher delay and reduced fairness compared to WFQ.

Chapter 3

Analysis of Virtual Time Complexity in Weighted Fair Queuing

3.1 Introduction

It was generally accepted that GPS calculations have $O(N)$ complexity [8, 12], without an actual proof of how one can achieve the worst case scenario in the simple case of no simultaneous arrivals. It has been mentioned previously in [6] that, if a link is shared by up to N active sessions, then updating the virtual time can incur a computation on a substantial subset of the N sessions or queues, which requires $O(N)$ time complexity. The $O(N)$ complexity was attributed to the so-

called iterated deletion [18] problem, and was the main reason why ideal WFQ has been replaced in practice with simplified approximations such as self-clocked [12] and start-time [16] WFQ algorithms. In this chapter, two fundamental theorems are presented that show that the $O(N)$ complexity for updating the virtual time in a WFQ scheduler with N sessions is caused mainly by simultaneous departures of packets, not by iterated deletion. Iterated deletion is caused by an "avalanche" of consecutive, but not simultaneous, departures that incur more departures due to increments in available bandwidth from idling sessions. Iterated deletion potentially leads to large numbers of consecutive departures within a time period. The number of departures is, however, a function of the resolution of the timestamp and the scheduler clock. Therefore, the number of consecutive departures within a time period can be made arbitrarily small, by using a finer-resolution virtual time update operation. In real software implementations, iterated deletion is avoided by considering only departures that have the same timestamp. In the case where iterated deletion is unavoidable, our proposed MW-WFQ algorithm can solve this problem as will be discussed in Chapter 4. On the other hand, the problem of simultaneous timestamps can not be solved by any increase in the time resolution of virtual time update. Essentially, all equal timestamps must be processed during a single virtual time update operation. In this chapter, a proof which shows that this is the real cause of the $O(N)$ complexity for virtual time update will be presented. Also, it will be shown that this is a fundamental property of WFQ that holds even under

the most severe restrictions, viz. all packets arrive serially to the scheduler, and the input bit-rate is equal to the output bit-rate.

3.2 GPS Revisited

As previously mentioned, WFQ simulates GPS in the background to produce timestamps for newly arriving packets. The timestamp is a sum of two components, a constant part and a variable part. The constant part is the ratio of the length of an arriving packet to its session link share (i.e. $\frac{L_i}{r_i}$). The variable part is the virtual start time which is the virtual time at which the packet will begin service under GPS. If a packet arrives in GPS to an idle session, then its virtual start time is simply the virtual time at arrival. If, on the other hand, a packet arrives to a backlogged session in GPS, then its virtual start time is exactly the virtual finish time of the previous packet which is simply the timestamp of the previous packet. Therefore, we do not need to calculate the virtual time for packets arriving to backlogged sessions in GPS. Only newly backlogged sessions in GPS require knowledge of the virtual time at the time of packet arrival. Note that we need to keep track of the set of backlogged sessions at all times in GPS to calculate the correct timestamps for WFQ.

Recall that the virtual time function in GPS is a piecewise linear function of time whose value at the start of a busy period equals zero and whose slope changes

according to:

$$\frac{\sum_{j=1}^{N} \alpha_j}{\sum_{i \in B(t)} \alpha_i} \tag{3.1}$$

where $B(t)$ is the set of backlogged sessions at time t and N is the total number of sessions. There are only two events that affect the slope of the virtual time function, one is the arrival of a packet to an idle session and the other is the departure of the last packet of a session (after which the session becomes idle). In between these two events, the slope of the virtual time remains fixed, because the set of backlogged sessions is fixed. The question now becomes, how frequent are these two events?

Before answering this question, we must make certain assumptions on the arrival process to a router's output link. In a router with m inputs, if all inputs simultaneously forward packets to the same output link, then these packets enter the output scheduler in a certain order. In other words, packet arrivals are serialized so that the scheduler sees arriving packets one at a time (Figure 3.1). This packet "serialization" process introduces a small fixed delay in a packet path that can be easily accounted for in delay calculations. Furthermore, since in many switch and router designs, an output buffer usually runs at several times the input link speed, therefore the fixed serialization delay becomes insignificantly small. Also, the maximum number of session arrivals to a router in a sufficiently small interval of time is at most m (this interval can be taken as the transmission time of the smallest packet of all the sessions).

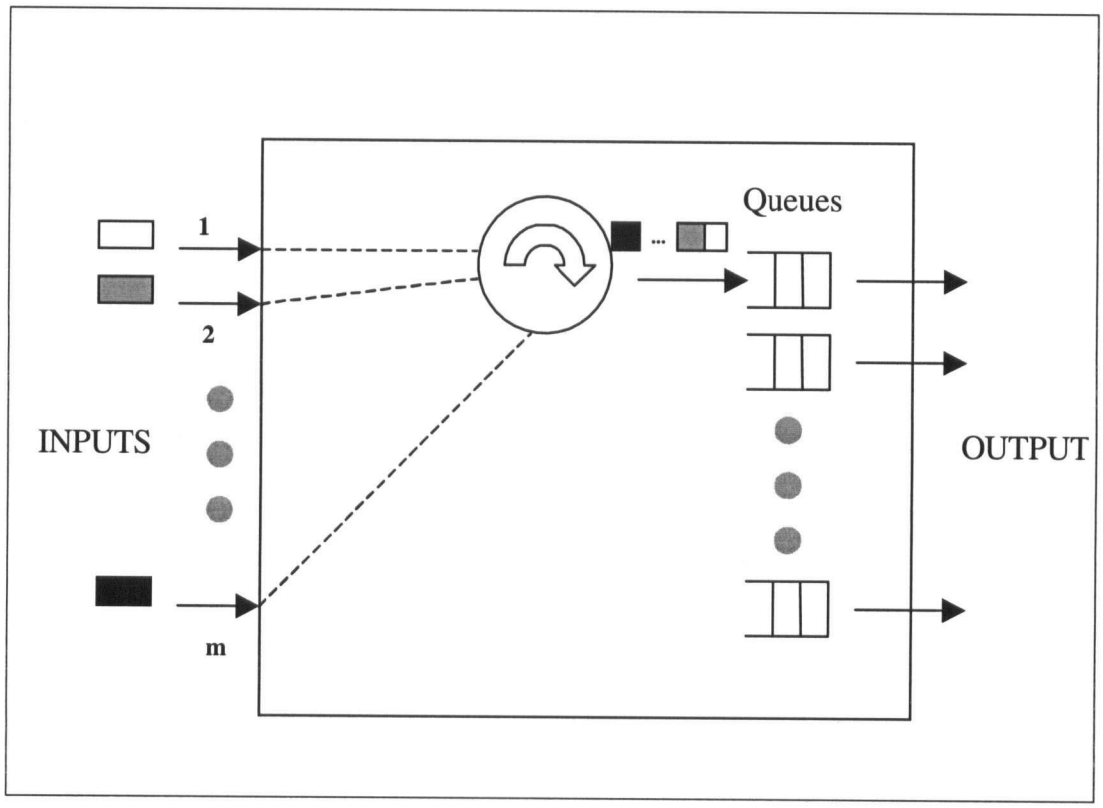

Figure 3.1: A model for the serialization of inputs that are destined to the same output.

Although the departure of the last packet of a backlogged session in GPS (henceforth, referred to as the event of a newly idled session) affects the slope of virtual time, we only need to consider this event when the next arrival to an idle session in GPS takes place. This is because, we only need the virtual time when calculating the timestamp of newly backlogged sessions. However, we need to remember the time at which the event of newly idle sessions took place. Therefore, the main events in our analysis will be arrivals to idle sessions in GPS. Although, only departure of the last packet of a session affects the slope of virtual time, we will consider any departure to represent an event that can affect the slope of virtual time. Consequently, we are concerned with only two types of events, arrivals to idle sessions and departures from GPS. When any of these events occurs, we need to recompute the value of virtual time. To do this, we start with the value of virtual time at the previous event and modify the slopes according to the two types of events. Initially, when the scheduler busy period starts and time is set to zero, the first packet to arrive at the output buffer will be an event of a newly backlogged session. Because the virtual time starts initially with the value zero, the initial timestamp will simply equal the constant part which is the ratio of packet length to link share (which is $\frac{L_i^k}{r_i}$ in equation 2.5). If the next arrival to the output buffer is for the same session, we can simply use the timestamp of the first packet. If, on the other hand, the second arrival is for a different session, we need the value of the virtual time at the time of packet arrival to the output buffer. So the problem of determining the time stamp

of a packet is reduced to the question of what happens to the virtual time between two consecutive packet arrivals to two idle sessions.

After a packet arrival to an idle session and before the next arrival to an idle session, two possible scenarios could happen. The first is that no packet departure takes place. The second is that one or more departures take place. In the first case, the set of backlogged sessions remains fixed between the two arrivals to the idle sessions. In the second case, the set of backlogged sessions may change between the event points. As an example, assume that we have two time instances, τ_1 and τ_2, where consecutive events of newly backlogged sessions take place. If no departures occur in between times τ_1 and τ_2, we have the virtual time case shown in Figure 3.2. When there are two departure events at times η_1 and η_2, between the two arrivals (events τ_1 and τ_2) resulting in newly idled sessions, we have the virtual time case depicted in Figure 3.3.

In Figure 3.2, during the interval (τ_1, τ_2) the set of backlogged sessions remains fixed and the slope remains constant. However, in Figure 3.3, the slope changes in each of the three intervals shown in bold. Note that the slope of the virtual time in the interval (τ_1, η_1) is the same as that of Figure 3.2 in the interval (τ_1, τ_2). At time η_1 some of the sessions that were backlogged in the interval (τ_1, η_1) become idle and are removed from the set of backlogged sessions. This causes the slope of the virtual time to increase in the interval (η_1, η_2). Similarly, some sessions that were backlogged in the interval (η_1, η_2) become idle, causing a further increase in

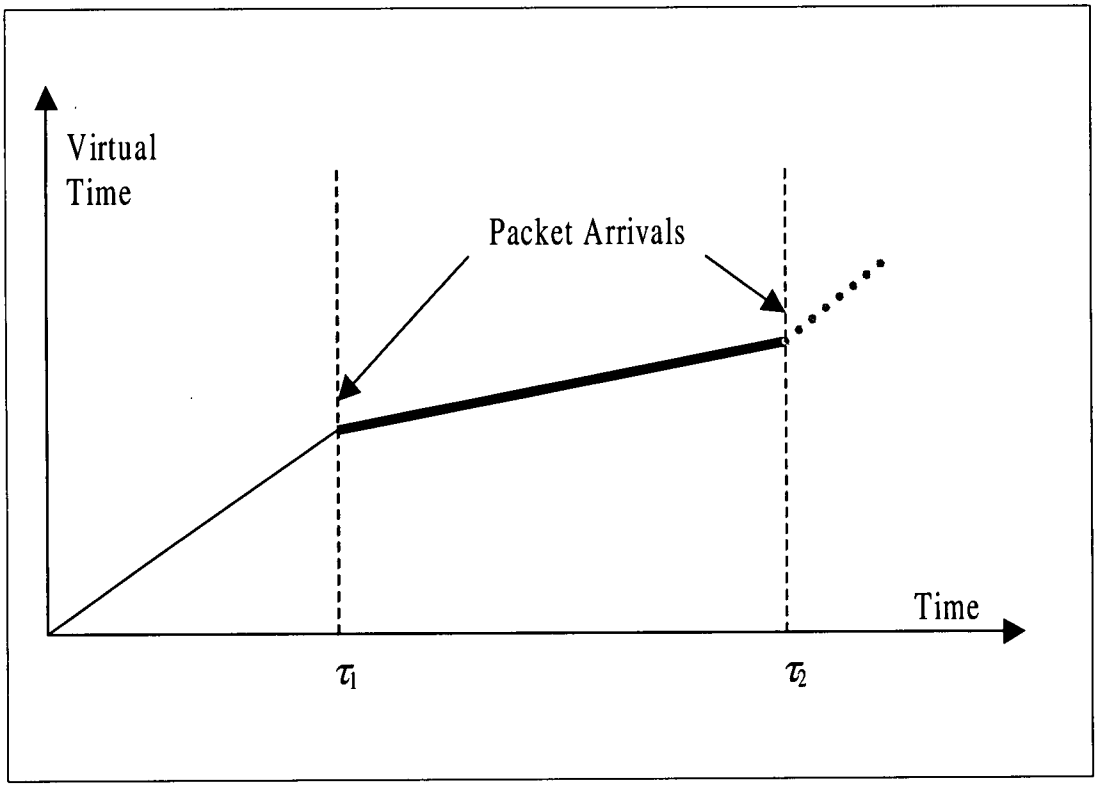

Figure 3.2: Virtual time function between two consecutive newly backlogged session events τ_1 and τ_2.

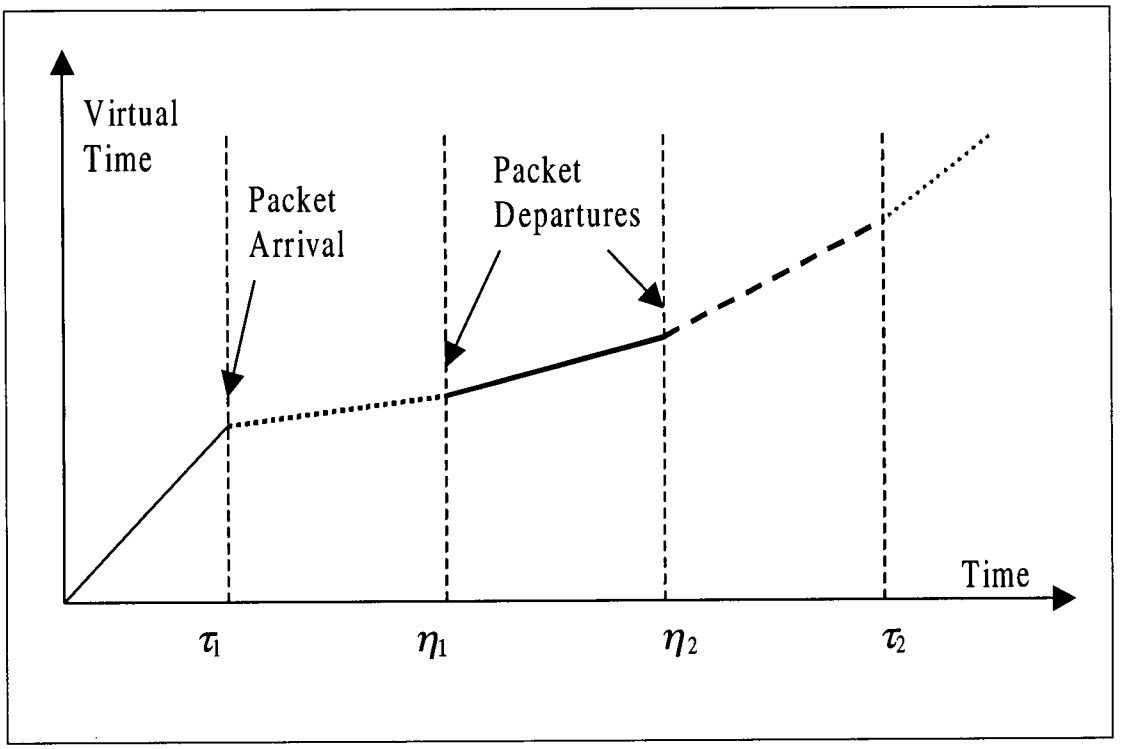

Figure 3.3: Virtual time function between two consecutive newly backlogged session events τ_1 and τ_2 when two newly idled departure events occur at times η_1 and η_2.

the slope of the virtual time in the interval (η_2, τ_2).

3.3 The Problem of Simultaneous Departures

In WFQ the most computationally expensive operation is maintaining the GPS virtual time function. The value of the virtual time function is inversely proportional to the sum of shares of backlogged sessions at any given instance of time. The set of backlogged sessions can change drastically from one instance of time to the next. This could happen due to either one of two reasons: *simultaneous arrivals* or *simultaneous departures* of packets to the scheduler.

Simultaneous arrivals can be the result of packets arriving from different input ports such that they are all destined to the same output link. If these arrivals are from sessions that were previously idle in GPS, then the slope of the virtual time will change after these packets arrive. Simultaneous departures are the result of several packets finishing their service in GPS at the same time. When this happens, it means that the value of the virtual time function in WFQ has just exceeded the timestamp value of these packets (that have equal timestamps). If some of these simultaneous departures cause some sessions to become idle, then this will cause the slope of the virtual time function to change. The amount of change in the slope of the virtual time in WFQ is controlled by the frequency of occurrence in GPS of both arrivals to idle sessions and equal finish times (departures).

It may be argued that if the input port traffic is properly multiplexed, such that no more than a single arrival takes place at a given time instance, then that would eliminate the simultaneous arrival problem. This type of "serialization" of input traffic does not solve the second problem of equal timestamps, i.e. equal departure times according to GPS. It was generally accepted that GPS calculations have $O(N)$ complexity, without an actual proof of how the worst-case complexity can be achieved in the simple case of no simultaneous arrivals to the scheduler. In fact, we will show in this work that WFQ can produce as many equal timestamp packets as the total number of active sessions. We will prove that the case of equal timestamps can occur even when the input and output link speeds are equal.

Now we present two fundamental theorems that establish sufficient conditions for WFQ to have a large number of packets with equal timestamps during a busy period, and assuming no simultaneous arrivals. *Theorem 1* shows that it is possible to have up to N equal-timestamp packets starting at the beginning of a busy period of a WFQ scheduler. *Theorem 2* shows that at an arbitrary point during the busy period of a WFQ scheduler, it is possible to have equal-timestamp packets equal to the number of idle sessions at that instance in time. In both Theorems, a relationship between the packet length and its session share and those of other sessions is shown. We can use *Theorem 1* and *Theorem 2* to create real scenarios where we can have up to N equal timestamps at different points in time.

3.3.1 Assumptions and Terminology

In the following two theorems we will find sufficient conditions under which WFQ will produce equal timestamps. Although WFQ can produce equal timestamps without them, we will make a few assumptions to help us find simple closed form expressions for the relationship between packet length and session share.

In these two theorems, packets arrive in a back-to-back manner, with no inter-packet gaps, and belong to unique sessions. The first packet arrives at the start of the busy period. In the second theorem, we assume that the busy period has already started. In this case, we are interested in generating equal timestamps for packets arriving at or after that moment in time. We also assume in both theorems that the input link is at least as fast as the output link.

Let t_i be the arrival time of the i^{th} packet to the GPS system such that its session number is also i, L_i is the length of the i^{th} packet, C_{in} is the input rate, and C is the output link rate such that $C_{in} \geq C$. Assume that $t_1 = 0$, $L_1 = L$, and assume that packets arrive in back-to-back manner with no inter-packet gaps, i.e.

$$t_n - t_{n-1} = \frac{L_n}{C_{in}}$$

Assume also, that at time $t = 0$, the virtual time $V(0) = 0$ and that the share of session i is α_i, where $\alpha_1 = 1, \alpha_i > 0, i \geq 2$.

Note that the above assumptions imply that only one packet arrives to each

session and therefore all packet arrivals are to idle (empty) session queues.

3.3.2 Theorem 1

Assuming that m packets arrive to m idle session queues, one packet to each queue (indexed from 1 to m), such that the packet lengths satisfy the relation:

$$\frac{L_n}{L_{n-1}} = \beta \frac{\alpha_n}{\alpha_{n-1}} \frac{\sum_{i=1}^{n-1} \alpha_i}{\beta \sum_{i=1}^{n-1} \alpha_i + \alpha_n}, \quad 2 \leq n \leq m \leq N \tag{3.2}$$

where $\beta = C_{in}/C$, then all the m packets will have the *same* timestamp value as that of the first packet.

Proof: According to the assumptions, the guaranteed rate r_i of session i according to GPS is $r_i = \alpha_i r$ where $r = C/\sum_{i=1}^{N} \alpha_i$. When packet 1 arrives at time $t_1 = 0$, it receives a timestamp equal to

$$TS(t_1) = TS(0) = V(0) + \frac{L_1}{\alpha_1 r} = 0 + \frac{L}{r} = \frac{L}{r}$$

(Since $\alpha_1 = 1$ and $L_1 = L$). Using the fact that, between times t_{n-1} and t_n only sessions 1, 2, $\cdots$ $n-1$ are active, we conclude that the virtual time slope during the same period of time is

$$\frac{C}{r \sum_{i=1}^{n-1} \alpha_i}$$

Therefore, we have:

$$V(t_n) = V(t_{n-1}) + \frac{C}{r\sum_{i=1}^{n-1}\alpha_i}(t_n - t_{n-1}) \tag{3.3}$$

From which we get:

$$V(t_n) = V(t_{n-1}) + \frac{C}{r\sum_{i=1}^{n-1}\alpha_i}(\frac{L_n}{C_{in}})$$

$$V(t_n) = V(t_{n-1}) + \frac{L_n}{r\beta\sum_{i=1}^{n-1}\alpha_i} \tag{3.4}$$

Since each packet arrives to an idle session, the timestamp of the *nth* packet is:

$$TS(t_n) = V(t_n) + \frac{L_n}{\alpha_n r} \tag{3.5}$$

Substituting 3.4 into 3.5 we get:

$$TS(t_n) = V(t_{n-1}) + \frac{L_n}{r\beta\sum_{i=1}^{n-1}\alpha_i} + \frac{L_n}{\alpha_n r} = V(t_{n-1}) + \frac{L_n}{r}(\frac{1}{\beta\sum_{i=1}^{n-1}\alpha_i} + \frac{1}{\alpha_n})$$

$$TS(t_n) = V(t_{n-1}) + \frac{L_n}{r}(\frac{\alpha_n + \beta\sum_{i=1}^{n-1}\alpha_i}{\alpha_n\beta\sum_{i=1}^{n-1}\alpha_i}) = V(t_{n-1}) + \frac{\alpha_{n-1}L_n}{\alpha_{n-1}r}(\frac{\alpha_n + \beta\sum_{i=1}^{n-1}\alpha_i}{\alpha_n\beta\sum_{i=1}^{n-1}\alpha_i}) \tag{3.6}$$

Substituting 3.2 into 3.6 we get:

$$TS(t_n) = V(t_{n-1}) + \frac{L_n}{\alpha_{n-1}r}\frac{L_{n-1}}{L_n} = V(t_{n-1}) + \frac{L_{n-1}}{\alpha_{n-1}r} = TS(t_{n-1}) \tag{3.7}$$

Using induction, it is simple to show that all the timestamps will be equal to $\frac{L}{r}$ which is the timestamp of the first packet, i.e.

$$TS(t_n) = TS(t_{n-1}) = \cdots = TS(t_2) = TS(t_1) = \frac{L}{r}$$

The above proof assumed that when a packet arrives, all the previous packets are still in the GPS system. To prove that this is indeed the case, we will show that the arrival time t_n is small enough to prevent all previous packets in GPS from departing. We do this by showing that t_{finish}, the predicted finish time at time t_{n-1} of all packets in the GPS system which have the same timestamp L/r, is greater than t_n and therefore the nth packet will arrive and further increase the finish time of these $n-1$ equal timestamp packets. In calculating t_{finish} we assume that the slope of the virtual time does not change in the interval (t_{n-1}, t_{finish}).

$$\begin{aligned} t_{finish} - t_{n-1} &= \frac{(\frac{L}{r} - V(t_{n-1}))}{\text{slope of virtual time in interval } (t_{n-1}, t_{finish})} \\ &= \frac{\frac{L}{r} - V(t_{n-1})}{\frac{C}{r\sum_{i=1}^{n-1}\alpha_i}} = \frac{\sum_{i=1}^{n-1}\alpha_i}{C}(L - rV(t_{n-1})) \end{aligned}$$

Using the fact that the timestamp of the nth packet is L/r, and using equation 3.7, we get

$$rV(t_{n-1}) = L - \frac{L_{n-1}}{\alpha_{n-1}} \Rightarrow L - rV(t_{n-1}) = \frac{L_{n-1}}{\alpha_{n-1}} = L_n(\frac{1}{\alpha_n} + \frac{1}{\beta\sum_{i=1}^{n-1}\alpha_i})$$

And we get:

$$\begin{aligned} t_{finish} - t_{n-1} &= \frac{L_n}{C}\left(\frac{\sum_{i=1}^{n-1}\alpha_i}{\alpha_n} + \frac{1}{\beta}\right) \\ &= \frac{L_n}{C}\frac{\sum_{i=1}^{n-1}\alpha_i}{\alpha_n} + \frac{L_n}{C\beta} \\ \Rightarrow t_{finish} - t_{n-1} &> \frac{L_n}{C\beta} \\ \Rightarrow t_{finish} &> t_{n-1} + \frac{L_n}{C\beta} \\ &> t_{n-1} + \frac{L_n}{C_{in}} \\ &> t_n \end{aligned}$$

therefore, $t_{finish} > t_n$.

Theorem 1 proved that it is possible to have up to N simultaneous departures in GPS. The following Corollary shows that even when the input rate is equal to the output rate and packet arrivals are serialized, a GPS system can still have up to N simultaneous departures.

3.3.3 Corollary 1

Given the same assumptions of *Theorem 1* and assuming that m packets arrive to m idle session queues, one packet to each queue (1 to m), such that $L > L_n, 1 < n \leq m$ and the packet lengths and shares are related by:

$$\alpha_n = \gamma_n \Pi_{i=2}^{n-1}(\gamma_i + 1),\ \beta = 1, 3 \leq n \leq m,\ and\ \alpha_2 = \gamma_2 \tag{3.8}$$

or equivalently,

$$\gamma_n = \frac{\alpha_n}{\sum_{i=1}^{n-1} \alpha_i},\ \beta = 1, 2 \leq n \leq m \tag{3.9}$$

where $\beta = \frac{C_{in}}{C}$ and $\gamma_n = (\frac{L}{L_n} - 1)^{-1}$, then all the m packets will have the *same* timestamp value as that of the first packet.

Proof: In 3.2, by assuming $\beta = 1$:

$$\frac{L_n}{L_{n-1}} = \frac{\alpha_n}{\alpha_{n-1}} \frac{\sum_{i=1}^{n-1} \alpha_i}{\sum_{i=1}^{n} \alpha_i}, \quad where\ \beta = 1$$
$$\frac{L_n}{L_{n-2}} = \frac{L_n}{L_{n-1}} \frac{L_{n-1}}{L_{n-2}} = (\frac{\alpha_n}{\alpha_{n-1}} \frac{\sum_{i=1}^{n-1} \alpha_i}{\sum_{i=1}^{n} \alpha_i})(\frac{\alpha_{n-1}}{\alpha_{n-2}} \frac{\sum_{i=1}^{n-2} \alpha_i}{\sum_{i=1}^{n-1} \alpha_i})$$

And:

$$\frac{L_n}{L_{n-2}} = \frac{\alpha_n}{\alpha_{n-2}} \frac{\sum_{i=1}^{n-2} \alpha_i}{\sum_{i=1}^{n} \alpha_i}$$

Hence by induction:

$$\frac{L_n}{L_1} = \frac{L_n}{L} = \frac{\alpha_n}{\alpha_1} \frac{\sum_{i=1}^{1} \alpha_i}{\sum_{i=1}^{n} \alpha_i} = \frac{\alpha_n}{\alpha_1} \frac{\alpha_1}{\sum_{i=1}^{n} \alpha_i} = \frac{\alpha_n}{\sum_{i=1}^{n} \alpha_i} \tag{3.10}$$

From 3.10:

$$\alpha_n = \frac{L_n}{L}(\sum_{i=1}^{n-1} \alpha_i + \alpha_n) \Rightarrow \alpha_n = \frac{\frac{L_n}{L} \sum_{i=1}^{n-1} \alpha_i}{1 - \frac{L_n}{L}}$$

And:

$$\alpha_n = \frac{\sum_{i=1}^{n-1} \alpha_i}{\frac{L}{L_n} - 1}, \quad \beta = 1, n \geq 2 \tag{3.11}$$

Substituting $\gamma_n = (\frac{L}{L_n} - 1)^{-1}$ in Equation 3.11, Equation 3.9 is obtained:

$$\alpha_n = \gamma_n \sum_{i=1}^{n-1} \alpha_i, \quad \beta = 1, n \geq 2$$

Using induction:

$$\begin{aligned}
\alpha_n = \gamma_n(\alpha_{n-1} + \sum_{i=1}^{n-2} \alpha_i) = \gamma_n(\gamma_{n-1} \sum_{i=1}^{n-2} \alpha_i + \sum_{i=1}^{n-2} \alpha_i) \\
= \gamma_n(\gamma_{n-1} + 1) \sum_{i=1}^{n-2} \alpha_i \\
= \gamma_n(\gamma_{n-1} + 1)(\alpha_{n-2} + \sum_{i=1}^{n-3} \alpha_i) = \gamma_n(\gamma_{n-1} + 1)(\gamma_{n-2} + 1) \sum_{i=1}^{n-3} \alpha_i \\
= \gamma_n(\gamma_{n-1} + 1)(\gamma_{n-2} + 1)(\gamma_{n-3} + 1) \cdots (\gamma_2 + 1)\alpha_1 \\
= \gamma_n(\gamma_{n-1} + 1)(\gamma_{n-2} + 1)(\gamma_{n-3} + 1) \cdots (\gamma_2 + 1)
\end{aligned}$$

Which is exactly Equation 3.8.

3.3.4 Examples

Example 1: Let us look at the case when $\beta = \frac{C_{in}}{C} = 1$. Assuming that the packet lengths are as follows: $L_1 = L$ and $L_i = \frac{1}{2}L, \quad i \geq 2$. This means that

$\gamma_n = (\frac{L}{L_n} - 1)^{-1} = 1, \ \ n \geq 2$ and from 3.8 we have $\alpha_2 = \gamma_2 = 1$ and for $n \geq 3$:

$$\alpha_n = \gamma_n \Pi_{i=2}^{n-1}(\gamma_i + 1) = \Pi_{i=2}^{n-1}(2) = 2^{n-2}$$

Thus, if shares are chosen as follows:

$$\alpha_1 = \alpha_2 = 1, \alpha_3 = 2, \alpha_4 = 4, \alpha_5 = 8, \alpha_6 = 16 \cdots$$

then all the arriving packets receive the same timestamp in GPS.

Example 2: Consider the case when $\beta = \frac{C_{in}}{C} = 1$ and assume that all the sessions have the same share α. Then, using Equation 3.9

$$\gamma_n = \frac{1}{\frac{L}{L_n} - 1} = \frac{\alpha_n}{\sum_{i=1}^{n-1} \alpha_i} = \frac{1}{n-1}, \ \ n \geq 2$$

which means

$$\frac{L}{L_n} = n, \ \ n \geq 2$$

Thus, if packet lengths are chosen as follows:

$$L_1 = L, L_2 = \frac{1}{2}L, L_3 = \frac{1}{3}L, L_4 = \frac{1}{4}L, L_5 = \frac{1}{5}L, L_6 = \frac{1}{6}L \cdots$$

then all the arriving packets receive the same timestamp in GPS.

To complete the theoretical analysis of virtual time the following theorem is presented which shows that at an arbitrary point during the busy period of a WFQ scheduler, it is possible to have many packets from different sessions with equal timestamps. Furthermore, the number of such packet can equal to the number of idle sessions at that instance in time.

Before stating the theorem, some assumptions are made; (a) the system consists of N sessions having shares $\alpha_i, i = 1, 2, \cdots, N$, such that at time t, sessions $1, 2, \cdots, k$ are backlogged while sessions $k+1, k+2, \cdots, N$ are idle at time t^-, (b) a single packet arrives to each of the idle sessions in back-to-back fashion, starting at time t, and in order of increasing session number. Let the arrival times of these packets be $t_{k+1}, t_{k+2}, \cdots, t_N$, respectively, and $t_{k+1} = t$ such that these arrivals have lengths $L_i, i = k+1, k+2, \cdots, N$. Then $t_{i+1} - t_i = \frac{L_{i+1}}{C_{in}}, i = k, k+1, \cdots, N-1$.

3.3.5 Theorem 2

If none of the backlogged sessions at time t^-, namely sessions $1, 2, \cdots, k$, become idle during the interval $[t_{k+1}, t_N)$[1], then by choosing the lengths $L_{k+1}, L_{k+2}, \cdots, L_N$

[1]Note that sessions which are backlogged at time t^- but are idle at time t^+ are not included in the set of sessions $1, 2, \cdots, k$.

to satisfy

$$\frac{L_n}{L_{n-1}} = \beta \frac{\alpha_n}{\alpha_{n-1}} \frac{\sum_{i=1}^{n-1} \alpha_i}{\beta \sum_{i=1}^{n-1} \alpha_i + \alpha_n}, \quad n = k+2, k+3, \cdots, N \tag{3.12}$$

the timestamps of the packets belonging to sessions $k+1, k+2, \cdots, N$ will all be equal to $V(t) + \frac{L_{k+1}}{\alpha_{k+1} r}$ where $V(t)$ is the virtual time at time t and $r = \frac{C}{\sum_{i=1}^{N} \alpha_i}$.

Proof: The timestamp of the packet arriving at time $t_n, n = k+2, k+3, \cdots, N$ is

$$TS(t_n) = V(t_n) + \frac{L_n}{\alpha_n r} = V(t_{n-1}) + (t_n - t_{n-1})(\text{slope of virtual time at } t_{n-1}) + \frac{L_n}{\alpha_n r}$$

$$= V(t_{n-1}) + \frac{L_n}{C_{in}} \frac{C}{r \sum_{i=1}^{n-1} \alpha_i} + \frac{L_n}{\alpha_n r}$$

$$= V(t_{n-1}) + \frac{L_n}{r} [\frac{1}{\beta \sum_{i=1}^{n-1} \alpha_i} + \frac{1}{\alpha_n}]$$

$$TS(t_n) = V(t_{n-1}) + \frac{L_n}{\beta r \alpha_n \sum_{i=1}^{n-1} \alpha_i} [\beta \sum_{i=1}^{n-1} \alpha_i + \alpha_n] = V(t_{n-1}) + \frac{L_{n-1}}{\alpha_{n-1} r} = TS(t_{n-1})$$

by induction, we get that $TS(t_{k+1}) = TS(t_{k+2}) = \cdots = TS(t_N)$.

Note that in this proof it was assumed that the k sessions that are backlogged at time t, remain backlogged during the interval $[t_{k+1}, t_N)$ to justify that the virtual time slope during the interval $[t_{n-1}, t_n)$ is $\frac{C}{r \sum_{i=1}^{n-1} \alpha_i}$.

It can be proved that if the conditions of *Theorem 2* are true, then when a packet arrives at time t_n, the packet that arrived at time t_{n-1} has not yet departed. The proof is similar to that of *Theorem 1*. The real question is how to guarantee that

none of sessions $1, 2, \cdots, k$ become idle during the interval (t_{k+1}, t_N)? To answer this question, it is first necessary to understand what can cause one of the first k sessions to become idle during the interval (t_{k+1}, t_N)? It is easy to prove that this will happen if the largest timestamp of one of these k sessions has a value that lies between the value of virtual time at times t_{k+1} and t_N.

If none of the first k sessions becomes idle during the interval (t_{n-1}, t_n), then the slope of the virtual time decreases from $\frac{C}{r\sum_{i=1}^{n-1}\alpha_i}$ to $\frac{C}{r\sum_{i=1}^{n}\alpha_i}$ at time t_n for $n = k+1, k+2, \cdots, N$. Also, at time t_{k+1} the virtual time $V(t_{k+1})$ is less than the maximum timestamp of each of the k backlogged sessions. To know whether or not one of the sessions $1, 2, \cdots, k$ will become idle in (t_{k+1}, t_N), $V(t_N)$ is calculated on the assumption that none of the sessions becomes idle in (t_{k+1}, t_N). Since $TS(t_N) = TS(t_{k+1})$, we have that:

$$V(t_N) + \frac{L_N}{\alpha_N r} = V(t_{k+1}) + \frac{L_{k+1}}{\alpha_{k+1} r}$$
$$\Rightarrow V(t_N) = V(t_{k+1}) + \frac{L_{k+1}}{\alpha_{k+1} r} - \frac{L_N}{\alpha_N r}$$

Let the largest timestamp of all packets belonging to session i at time t be $TS_i^{max}(t)$. Define

$$\overline{TS}(t) = \min_{1 \le i \le k} \{TS_i^{max}(t)\}$$

If $\overline{TS}(t_{k+1}) > V(t_{k+1})$ and $\overline{TS}(t_{k+1}) \ge V(t_N)$, then the assumption will be valid. In

general, it is required that

$$\overline{TS}(t) > V(t), \quad t \in (t_{k+1}, t_N) \tag{3.13}$$

Condition 3.13 allows arrivals to sessions $1, 2, \cdots, k$ during the interval (t_{k+1}, t_N) and guarantees that all of these k backlogged sessions will remain backlogged until time t_N. One way to guarantee that 3.13 is satisfied is to choose either L_{k+1} or α_{k+1} such that

$$TS(t_{k+1}) \leq \overline{TS}(t_{k+1}) \tag{3.14}$$

but in general Equation 3.13 is less restrictive than Equation 3.14.

Chapter 4

A Scalable Minimal-Work Algorithm for Computing the Virtual Time in GPS

4.1 Introduction

This chapter introduces a novel algorithm called Minimum-Work Weighted Fair Queuing (MW-WFQ) [19] for implementing WFQ that eliminates the $O(N)$ computational complexity problem in standard WFQ implementations. The algorithm represents a significant advancement in per-flow scheduling and solves the long standing scalability problem that was associated with such algorithms. As a starting point, we will discuss how the standard WFQ algorithm is implemented. Once the prob-

lems with the standard WFQ implementation are understood, it will be shown how the algorithm deals with the $O(N)$ complexity problem.

Recall from the previous chapter that the $O(N)$ computational complexity can occur during the computation of the timestamp of a single packet, and it is thus the main cause of the WFQ scalability problem.

In this chapter, an implementation of a standard WFQ algorithm is first presented, then a novel Minimum-Work WFQ algorithm is proposed.

4.2 A Standard WFQ Implementation

In WFQ, the timestamp of a packet is determined upon arrival and based on whether or not it arrives to an idle session in GPS [7]. If the arrival is to a backlogged session, then the virtual time is not important in calculating the timestamp. If, on the other hand, the arrival is to an idle session, the value of virtual time at that instance must be computed. If describing how the virtual time function changes between two consecutive events of arrivals to idle sessions is possible, then calculating a packet's timestamp becomes easy. In what follows, a simple algorithm is proposed for calculating the virtual time between any two consecutive times where packets arrive to idle sessions. Let τ_i and τ_{i+1} denote the times of two consecutive newly backlogged session events, i.e. packet arrivals to two different idle sessions. Also, let $V(t)$ denote the virtual time at time t, and let TS_i be the smallest timestamp

of all the packets belonging to session i in GPS. Then the algorithm for calculating the virtual time can be formulated as follows:

Begin:

1. Calculate the minimum timestamp TS_{min} at time τ_i(the "+" after τ_i indicates that any departure have been considered first before calculating the minimum timestamp) :

$$TS_{min} = \min TS_j, \forall j \in B(\tau_i^+)$$

2. Calculate the finish time t_{finish} for this minimum timestamp as follows:

$$t_{finish} = \tau_i + [TS_{min} - V(\tau_i)] * \sum_{j \in B(\tau_i^+)} \overline{\alpha}_j$$

3. If $t_{finish} = \tau_{i+1}$ then $V(t_{finish}) = V(\tau_{i+1}) = TS_{min}$, done.

4. Else if $t_{finish} < \tau_{i+1}$, then all the sessions whose head packets have timestamps equal to TS_{min} will exit the GPS scheduler at time t_{finish}. Subsequently, it may be necessary to adjust the slope of the virtual time after time t_{finish} as a result of some sessions becoming idle. The virtual time at time t_{finish} becomes:

$$V(t_{finish}) = TS_{min}$$

5. Find the new value of minimum timestamp TS'_{min} at time t_{finish} after packets

exit the GPS scheduler:

$$TS'_{min} = \min TS_j, \forall j \in B(t^+_{finish})$$

6. Find the value of the new finish time t'_{finish} corresponding to this new minimum timestamp TS'_{min}:

$$t'_{finish} = t_{finish} + [TS'_{min} - V(t_{finish})] * \sum_{j \in B(t^+_{finish})} \overline{\alpha}_j$$

7. If $t'_{finish} < \tau_{i+1}$, then like in steps 3-6 packets will exit the GPS scheduler at this new finish time. The new finish time will have a virtual time equal to the new minimum timestamp:

$$V(t'_{finish}) = TS'_{min}$$

8. The minimum timestamp TS''_{min} and next finish time t''_{finish} are calculated for the new finish time t'_{finish}. This process is repeated until we reach a value of time (call it $\bar{t}$) such that the next calculated finish time is greater than or equal to τ_{i+1}.

9. If $\bar{t} < \tau_{i+1}$, we calculate the virtual time at time τ_{i+1} as follows:

$$V(\tau_{i+1}) = V(\bar{t}) + [\tau_{i+1} - \bar{t}] / \sum_{j \in B(\bar{t})} \overline{\alpha}_i$$

End

Note that this algorithm is able to find the value of virtual time at the next event of newly backlogged sessions given the value of virtual time at the previous event of newly backlogged sessions. The algorithm is initiated at time zero with virtual time set equal to zero. This algorithm can be used as a basis for implementing a standard WFQ scheduler. Figure 4.1 is a flow chart that shows such an implementation of WFQ which calculates the timestamp of an arriving packet based on this proposed algorithm.

In the flow chart of Figure 4.1 an arriving packet is given a timestamp equal to the virtual time at the instance it leaves the GPS scheduler. The chart has four main cycles. Cycle 1, which traverses the branches $\{8, 9, 5, 6, 3, 10, 12, 13\}$, corresponds to the case of a newly backlogged session event. Cycle 2, which traverses the branches $\{10, 11, 5, 6, 3\}$, corresponds to the case of an arrival to a backlogged session. Cycle 3, which traverses the branches $\{7, 8, 9, 5, 6\}$, corresponds to departures in GPS. Cycle 4, which traverses the branches $\{3, 4, 5, 6\}$, corresponds to the case of no arrivals and no departures in which time is simply advanced. There are only two calculations needed, one to find the finish time and the other to calculate the virtual time. Note

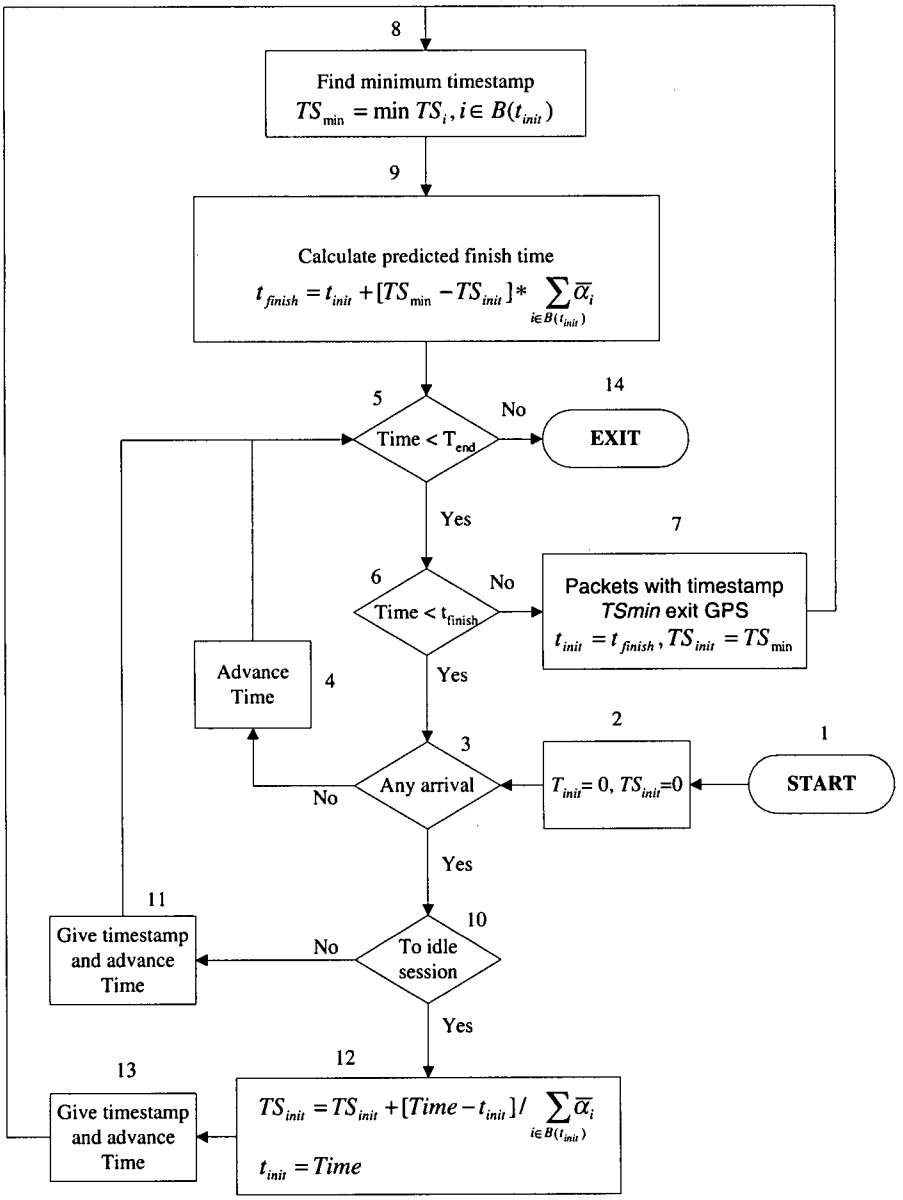

Figure 4.1: Flowchart of a standard WFQ implementation.

that the algorithm requires access to the minimum timestamp which requires the use of a suitable priority queue realization. The top of the routine begins with a minimum timestamp discovery among all the heads of the queues of backlogged sessions. Note that the routine begins at the "START" point.

4.3 GPS Simulation

To investigate the feasibility of GPS simulation it is necessary to address the following issues. The first issue is how frequent the two types of events are. This helps in determining the amount of time available to perform the computations between events. At the start of a busy period, there is a high likelihood that most of the arrivals will correspond to one of the two events, especially the event of newly backlogged sessions. As mentioned earlier, such an event incurs a heavy computation, but at the same time may require as little as a packet transmission time to be executed. The second issue is the time it takes to determine the minimum timestamp of all backlogged sessions. This step will be the crucial part in determining the execution time of the algorithm cycles. If this search time can be reduced then the cycles can be completed in time. It might be useful to keep a sorted list of timestamps so that we can determine the minimum timestamps quickly and be able to decide promptly the next departure time.

Up to this moment, only continuous time and discrete arrival processes were

considered. These arrival processes are discrete because a packet arrives only when its last bit reaches the output buffer. In a practical packet system, both time and arrivals are discrete. Therefore, a "scheduling clock" is needed during which the system reads input packets, calculates their timestamps and queues them appropriately. Note that we do not mean here the actual hardware clock driving the router or switch hardware. Rather, we mean a "task clock" that executes the task of scheduling regularly. The shorter the scheduling clock period, the more accurate the algorithm will be. This is due to the fact that an arrival that takes place during a clock cycle is not considered until the beginning of the next clock. In effect, this results in a delay in the packet transmission time which is proportional to the scheduling clock period. Ideally, a scheduling clock with infinitesimally small period is needed. This is not practical of course, but a suitable compromise can be found that adds a small delay to packet transmission times. Delay is not the only side-effect of a discrete clock; packet reordering is another. If two packets arrive one after the other during a clock period, they will both be treated as simultaneous arrivals. This may cause them to receive timestamps that order them upon transmission differently from what would happen if the clock were continuous. However, these effects result in very small delay and negligible "local" reordering of packets.

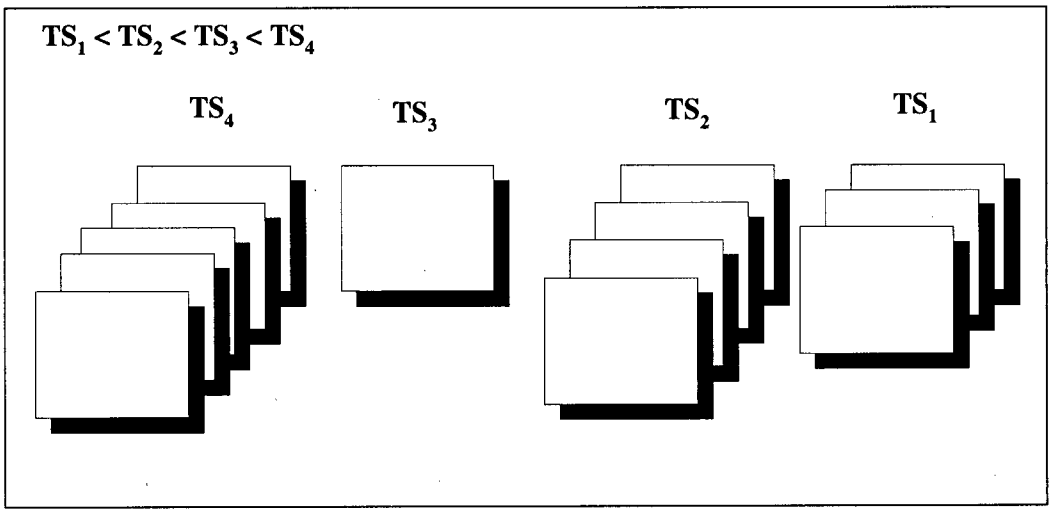

Figure 4.2: Timestamp groups.

4.4 Minimum-Work Weighted Fair Queuing

The algorithm of Figure 4.1 performs only two significant computations, one for computing the finish time (t_{finish}) and the other for computing the virtual time (TS_{init}). Both of these computations contain the quantity $\sum_{i \in B(t)} \overline{\alpha}_i$, which, for convenience, will be called the *Backlog Sum* at time t. Calculating this quantity requires accumulating the shares of all backlogged sessions at any given time. The naive approach of computing this value at packet arrivals incurs a computation time proportional to the number of backlogged sessions which can be quite long ($O(N)$ where N is the number of sessions sharing the output link). What is needed is a way of reducing the time it takes to calculate this sum. The key to the proposed solution is to build up the backlog-rate incrementally so that only a fixed number of operations is required to maintain this sum after any event. A data structure

that keeps all the timestamps in the system in sorted order (by increasing timestamp value) is proposed. Figure 4.2 depicts part of this data structure. The figure shows 13 packets with four distinct groups of packets sorted by increasing order of timestamps from TS_1 to TS_4. Note that each group may contain several packets with same timestamp TS_i. To reduce memory requirements, the data structure uses small records to represent packets. Each record will have a pointer to the packet it represents. The actual packet will remain in the packet memory. In a given group, the rightmost record is used to represent the first packet to arrive with a timestamp equal to the group timestamp. The packets are listed, within a group, in an increasing order of arrival. For example, the minimum timestamp group is TS_1 and contains three packets. The packet at the top is the first packet to arrive to the system with a timestamp equal to TS_1. The packet beneath it and to its left is the second packet to arrive to the system with a timestamp equal to TS_1. Because all the timestamps in the GPS system are enqueued in this sorted queue, two packets belonging to the same session will necessarily belong to different timestamp groups. In other words, all packets belonging to the same timestamp group must belong to different sessions.

To calculate the timestamps quickly, it is necessary to maintain the following three sums: One is the *Backlog Sum*, discussed before, which is the sum of the shares of all backlogged sessions at a given time. The second and third sums are local sums maintained for each timestamp group. The second sum is called *Share*

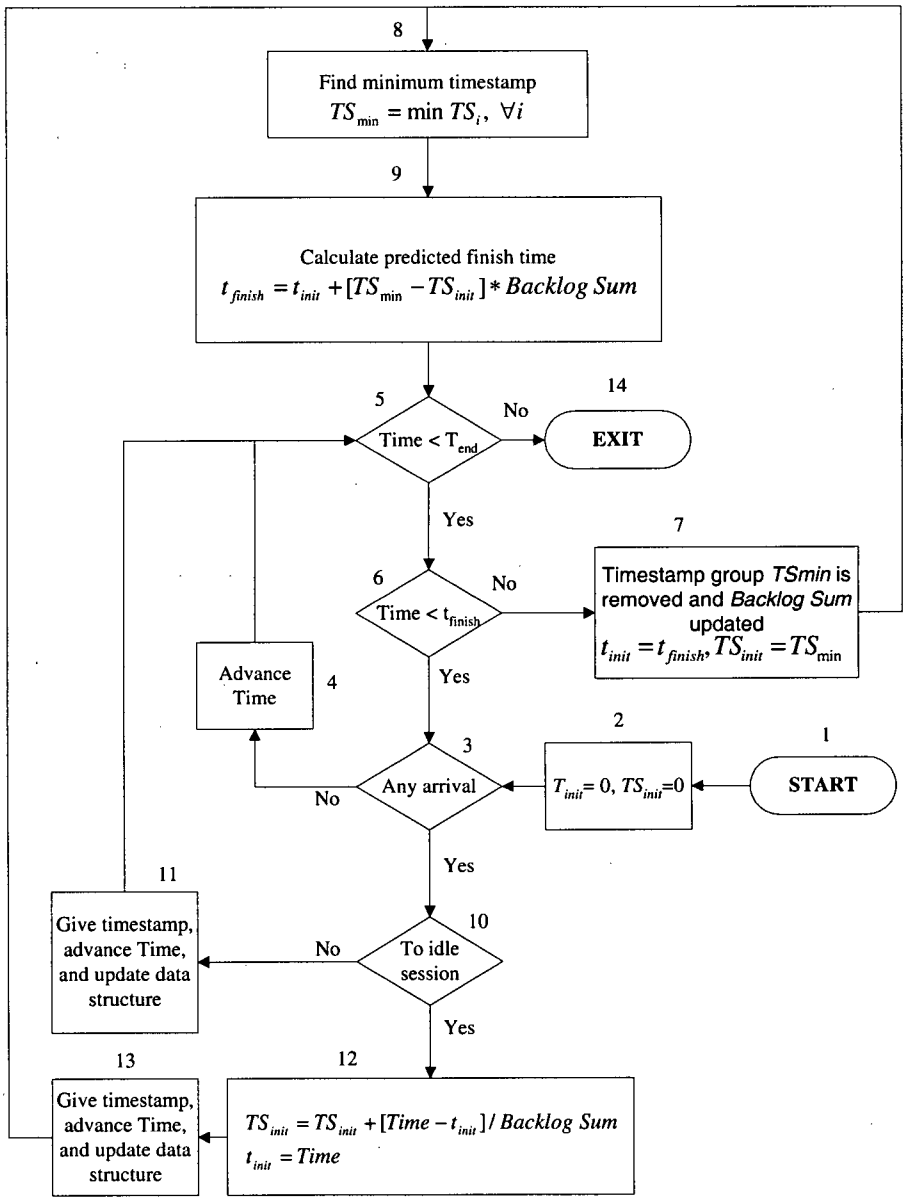

Figure 4.3: Flowchart of the MW-WFQ algorithm.

Sum and is the sum of the shares of all the sessions that have a timestamp equal to TS_i. The third sum, called *Next Sum*, is the sum of the shares of all the sessions that have packets with timestamp TS_i and which are not the last packets in their respective sessions (i.e. each of these sessions has packets with timestamp greater than TS_i). With *Backlog Sum* maintained for the entire system and *Share Sum* and *Next Sum* maintained for each timestamp group, an optimized algorithm for calculating WFQ timestamps can be defined as in Figure 4.3. This new algorithm is called Minimum-Work Weighted Fair Queuing or MW-WFQ. In this algorithm $\sum_{i \in B(t)} \overline{\alpha}_i$ is replaced with *Backlog Sum*. The main idea of the algorithm is the observation that, whenever a timestamp group TS_i leaves the GPS system, its *Next Sum* gives the sum of rates of all sessions that remain backlogged after their packets (with timestamp TS_i) leave the GPS system. This is equivalent to finding which sessions remain backlogged after the transmission of the packets with timestamp TS_i. The algorithm subtracts *Share Sum* and adds *Next Sum* to *Backlog Sum* to determine the new value of $\sum_{i \in B(t)} \overline{\alpha}_i$ after the departure of a TS group. Notice that all the fields in the data structure are updated every time a packet arrives to or leaves GPS. Therefore, our data structure allows incremental updates of all the values that are needed to compute the virtual and finish time.

The calculated timestamp of a backlogged or newly backlogged session may result in a new value of timestamp and hence a new timestamp group. This new value has to be queued in its proper place in the timestamp structure. This requires the use of

a priority queue structure. The delay encountered in inserting into a priority queue is a problem that is not limited to WFQ but is rather a problem with all sorted priority schedulers. In particular, all GPS emulations, such as WFQ, SCFQ, SPQ, VT, SPFQ and MD-SCFQ, face this problem. If this insertion into a sorted queue problem can be achieved in a constant small amount of time which is independent of the number of backlogged sessions i.e. O(1), then updating the timestamp structure can be done in $O(1)$ time. In the worst case, $O(\log N)$ is achievable using well-known balanced heap data structures. In that case, the implementation cost of WFQ is equivalent to any approximation of WFQ. Hence, there are no computational advantages to approximating WFQ using any technique such as SCFQ or MD-SCFQ. In the following section, further ways of reducing the implementation costs of WFQ are discussed.

4.5 Timing Issues in WFQ Implementations

Practical timing considerations when simulating a true GPS system have not been addressed adequately in the literature. Most of the literature deals with ways to approximate the virtual time function and does not cover the effects of discrete time on the implementation of WFQ or any of its variants. Because the proposed algorithm performs true GPS simulation, these timing issues must be carefully analyzed. One major issue is how GPS events relate to the scheduling clock on which

GPS is simulated. For example, GPS packet arrivals and departures can occur at arbitrary points in time. In a practical system, however, such events are normally synchronized with the beginning of a clock period (assuming a synchronous system). In that case, we do not allow "interrupts" of a scheduling cycle to take place. In this section, a detailed account of the impact of system timing on WFQ algorithm is provided. Implementations that use a variable clock vs. a fixed clock are also considered.

It has been shown in the previous section that maintaining the virtual time can be accomplished by keeping track of the three sums *Backlog Sum*, *Share Sum* and *Next Sum*. It is also mentioned that any practical realization of a packet scheduler must have a finite scheduling clock. This means that input and output events take place only at clock boundaries. It is, therefore, appropriate to assume that all inputs arriving during a clock period are delayed until the beginning of the next clock period. In other words, the decisions made about departures and arrivals are based on the value of time at the start of the scheduling clock period as follows:

At the beginning of a scheduling clock period, the departures are examined to see whether any timestamp group (or groups) should depart GPS. Observe that, within one clock period more than one timestamp group can depart the system. This is due to the fact that a timestamp group departs whenever its TS field is equal to virtual time. However, since the system is finite, it is possible that more than one timestamp exits GPS within the same clock period (iterated-deletion).

This occurs when the increasing value of virtual time during a clock period becomes equal to more than one timestamp queued in the timestamp queue. In this case, time is not advanced before clearing all the existing timestamp groups and obtaining the correct value of virtual time at the end of that clock period. As a result, the departure cycle may be executed more than once within a clock period. This may lead to the scheduling clock period becoming larger than originally assumed. It must be guaranteed, that no matter how many timestamp groups depart in a single clock period, the length of the cycle remains the same. Note that this problem does not exist in continuous-time GPS since only one timestamp group is eligible for departure at any given time. This iterated-deletion problem is easily solved by MW-WFQ by doing a simple modification to the basic algorithm to allow us to search for the group that will depart last. Once this group is identified, we can update the system in O(1) steps. In our patent document, further details of our algorithm to solve this potential problem are discussed [19].

After packet (group) departures are processed, the algorithm checks for arrivals. The appropriate timestamp is calculated and inserted into a sorted priority queue of timestamps. This part is the most computationally expensive in the algorithm and a good implementation of the priority queue leads to smaller clock periods and hence more accuracy. It has been assumed all along that there is at most a single arrival in a clock period. This assumption can be justified by ensuring that the clock is smaller than the smallest packet inter-arrival time. Note that the lower bound on

the inter-arrival time is the transmission time of the smallest possible packet length.

4.5.1 Fixed Clock vs. Variable Clock

Choosing the appropriate length for the scheduling clock is crucial to the proposed algorithm because only $O(1)$ operations are to be performed during one clock period. The scheduling clock duration can be either fixed or variable. To use a scheduling clock with fixed duration, two requirements need to be satisfied; (a)the length of the clock period is smaller than the smallest inter-arrival time between two packets, and (b)the period is long enough to enable the algorithm to finish all the calculations needed in that period. These calculations deal with departures and arrivals as described before. The longest of the two calculations is the departure, due to the fact that more than one timestamp group may depart in one clock period.

Another more efficient implementation of the WFQ algorithm is to have a variable clock period. The length of the period is the time it takes to fully execute an arrival or departure calculation. Hence, the cycle is started by finishing all the calculations involved in determining what timestamp groups should depart since the last cycle was executed. Since these calculations depend on the length of the previous cycle, we may end up with a longer or shorter current cycle. For example, if the previous cycle had no arrivals or departures and the present cycle has an arrival, then the present cycle can be longer than the previous one. As a result, the cycles will vary with time depending on the arrival and departure pattern. The variable

scheduling clock has the same two requirements as does the fixed-period scheduling clock.

4.5.2 Priority Queue Implementation

As mentioned previously, a good priority queue implementation leads to shorter algorithm cycles and hence faster scheduling clocks. This, in turn, leads to less scheduling delay for arriving packets since an arriving packet does not wait long till the start of the next system clock cycle.

The packet enqueue/dequeue from a priority queue is the most time consuming operation that has to be completed within one scheduling clock cycle, especially when the number of distinct timestamps is large (e.g. $N > 64000$). In fact, because all other operations require $O(1)$ time only, the time complexity of virtual time/timestamp computations are lower bounded by the time to enqueue/dequeue from a priority queue structure. Specifically, the time complexity of computing virtual time/timestamp following an arrival or departure event in GPS is $O(Q(M))$, where $Q(M)$ is the time required to enqueue/dequeue a packet (header) in a priority queue with M distinct TS groups. Now $Q(M)$ depends on the particular implementation of priority queue used. For example, a typical sequential realization of a priority with M elements can be based on a balanced heap data , with insert/delete time of $O(\log M)$. However, more efficient realizations based on a Calendar queue data structure [17] can result in $O(1)$ time access in most cases. Finally, efficient

hardware realizations of a priority queue based on systolic operation are proven to require $O(1)$ time for priority queue read (or delete),however, insert operations can require a longer time [20].

In our case, even a sequential $O(\log M)$ time access priority queue is adequate for attaining very high packet forwarding speeds. Indeed, a software-only realization can achieve a forwarding rate of a few hundred thousand packets per second on a typical 300MHz processor, based on a straightforward software implementation. Therefore, the proposed scheduler is able to forward packets at wire speed for high-speed links using software-only implementations.

4.5.3 Clock and Timestamp Selection

The success of the WFQ implementation depends on the correct choice of a scheduling clock period. Three factors influence the length of the scheduling clock. One of these factors is the amount of queuing delay we are willing to tolerate given that an arriving packet cannot be serviced except at the start of a scheduling clock. Basically, the longer the scheduling clock is, the longer a packet is delayed. The second consideration in choosing the scheduling clock period is the service order. With a longer clock, it is more likely that the actual service order will be different from the ideal service order in the continuous-time model. This is due to the fact that arrivals within a period are assumed to happen all at the start of the next period. Although one can keep track of the exact arrival time of a packet, it is difficult to calculate

the virtual time at the time of its arrival. To see why this is the case, consider the following scenario. When a packet arrives during a scheduling clock period in which the finish times are being calculated, that packet could be arriving to an idle session and hence can affect the finish time calculations. Although it is known that there is a single arrival at any given time, it is not known when that arrival will occur relative to the scheduling clock, and therefore it is assumed that such arrivals are aligned with the scheduling clock event. The scheduling clock represents the time it takes to update information about the virtual time. The proposed algorithm always starts by updating information pertaining to departures that took place during the past clock period, then it deals with arrivals in the present period.

The third factor that affects the length of the scheduling clock is the time it takes to execute the longest cycle in the algorithm. This time is dependent on the time it takes to update the timestamp data structure. The faster we can insert a record into this structure, the shorter the cycles of the algorithm are, and consequently the shorter the clock period can become. This reduction in the length of the clock period leads to a reduction in the number of simultaneous timestamp group departures. Therefore, we need to make sure that we choose the clock period to be short enough to finish the calcuations in time, but long enough to enable the calculation of multiple timestamp departures. If we want to guarantee that few timestamps are eligible at the same time, we can choose the timestamps from a set that keeps the distance between different timestamp values large enough to prevent them from becoming

eligible at the same time.

We have been assuming all along single arrivals in a period by forcing the period size to be smaller than the inter-arrival times. However, it is feasible to allow multiple packet arrivals in the same clock period. In this case, we need to calculate more than one timestamp. The value of virtual arrival time will be the same for all the packets arriving within the same clock period. If one of the packets belongs to an idle session, we compute the virtual time and give each packet a timestamp. After that, we start the departure calculations. The main problem with multiple arrivals is the need to insert more than one item into the timestamp structure. If such a delay is affordable, the system clock period can be made large enough to accept more than a single arrival.

The selection of timestamp accuracy has a strong impact on the implementation of WFQ. The more accurate the values of timestamps are, the less likely there will be multiple departures in a single cycle. However, this might be expensive to realize given the word length required to achieve good accuracy. Choosing a coarser accuracy for the timestamps, on the other hand, leads to a higher likelihood of multiple departures in a single clock period. Therefore, the choice of timestamp accuracy is a trade-off between having extensive calculations and better fairness properties.

4.5.4 WFQ Implementation Based on New Data Structure

Three entities are required for implementing the MW-WFQ algorithm. One entity is the TS group data structure, the second is the GPS session queues, and the third is the packet scheduler that schedules the output link according to the WFQ mechanism. The relationship between the three entities is shown in Figure 4.4. The GPS queues is only required for maintaining the state of a session, i.e., whether it is backlogged or idle. It is not necessary for every packet be accounted for in the GPS queues.

Upon arrival of a packet, a timestamp is calculated using the state of both the TS group data structure and the GPS session queues. Both the TS groups and GPS queues are interdependent. Once the timestamp of the arriving packet is determined, both the TS group and GPS queues are updated. The timestamp is also placed into the packet scheduler's priority queue. The scheduler then chooses for transmission the packet with the minimum timestamp.

The WFQ priority queue uses only values of timestamps and does not queue actual packets. In addition to the timestamp of a packet, the WFQ priority queue typically maintains a pointer to the position of the packet in the packet memory to enable immediate access to the packet once it becomes eligible for transmission. When a packet departs the WFQ scheduler, it can be deleted from the packet memory, as it serves no purpose for either the WFQ scheduler or the GPS simulation.

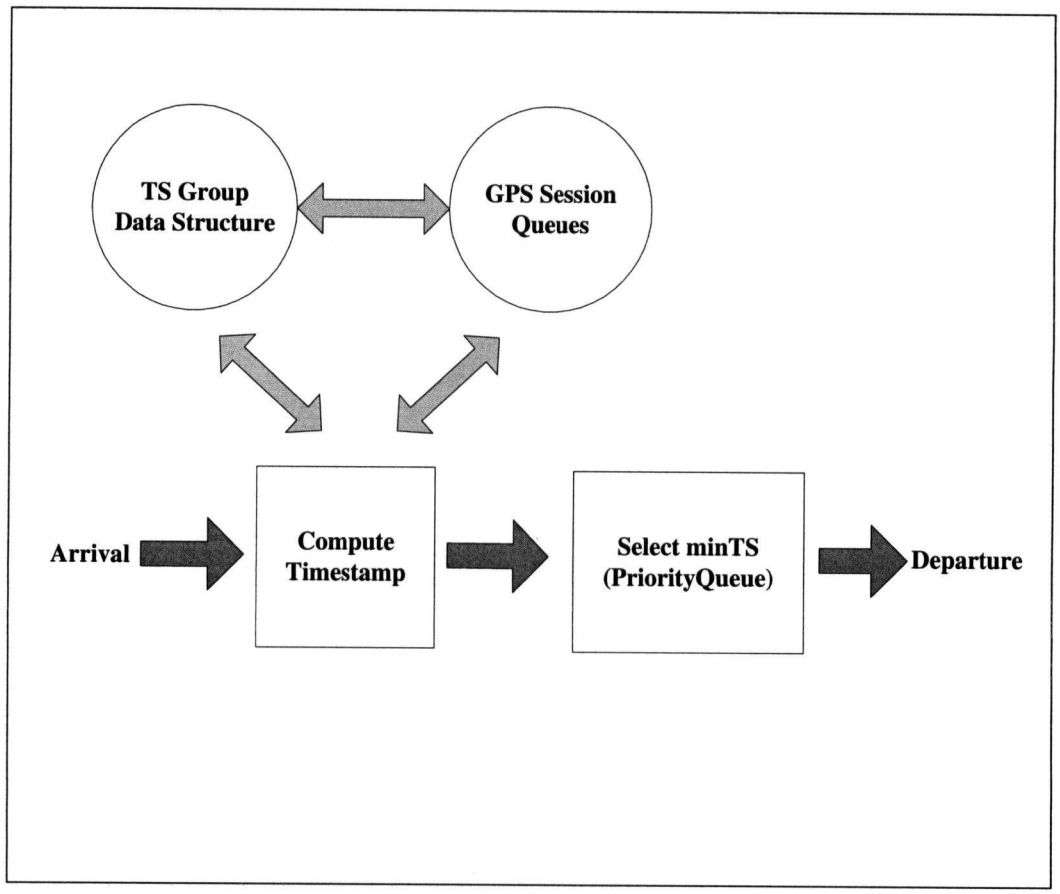

Figure 4.4: The components of a WFQ implementation.

4.5.5 WF^2Q Implementation

Worst-case Fair Weighted Fair Queueing (WF^2Q) is a modification of WFQ that improves the worst-case fairness of WFQ [21]. The worst-case fairness is an indication of how close the service a session receives in a packet scheduler is to that provided by GPS in any interval of time. To implement WF^2Q, we use a packet regulator that queues packets and only chooses for transmission the packet with the minimum timestamp among the set of all eligible packets. A packet is eligible if its virtual start time is greater or equal to the present value of virtual time. The virtual start time of a packet can be calculated from its timestamp by subtracting the ratio of packet length to product of server rate and session share:

Virtual start time = timestamp - Length / (rate of server * session share)

To implement WF^2Q, we can use the same implementation of WFQ with the addition of a regulator that verifies that a packet is eligible before allowing it to be transmitted. If a packet is not eligible, the next packet in the WFQ priority queue is checked for eligibility and so on.

4.5.6 Software Implementations of WFQ

Different C-language-based implementations of the scheduler have been written and tested. The tests have shown that the newly proposed WFQ implementation produces the correct output. Two different versions of the algorithm have been im-

plemented; one with a fixed system clock and the other with a variable one. The variable clock implementation will be discussed later in Chapter 5. In the fixed clock implementation, the purpose was to implement a WFQ simulator. This simulator accepts any arrival traffic pattern and produces the correct WFQ output to any degree of accuracy. This simulator uses a scheduling clock that synchronizes packet arrivals with the start of a clock period. The shorter the clock period is, the closer the simulations are to the ideal GPS scheduler. There are two parameters that are provided in addition to the scheduling clock period. One parameter is the time unit (or clock period) resolution, the other parameter is the timestamp resolution. The time unit resolution controls how accurate our time values are. The larger the value of the time unit resolution is, the more events are assumed to occur in the same time instance. Similarly, the timestamp resolution controls the degree to which two timestamp values are assumed to be equal with higher values, indicating a bigger range of values of timestamps taken to be equal.

In the following, we will show the effects of finite scheduling clock on the performance of our WFQ simulator in order to determine the optimum values for the scheduling clock period. In addition to the scheduling clock, we will investigate the effects of time unit and timestamp resolutions on the performance of the scheduler. To illustrate the effects of scheduling clock, time resolution and timestamp resolution, we present the following example [4]: Assume that we have two sessions having equal shares of an output link whose rate is 1. Let the arrivals to the scheduler be

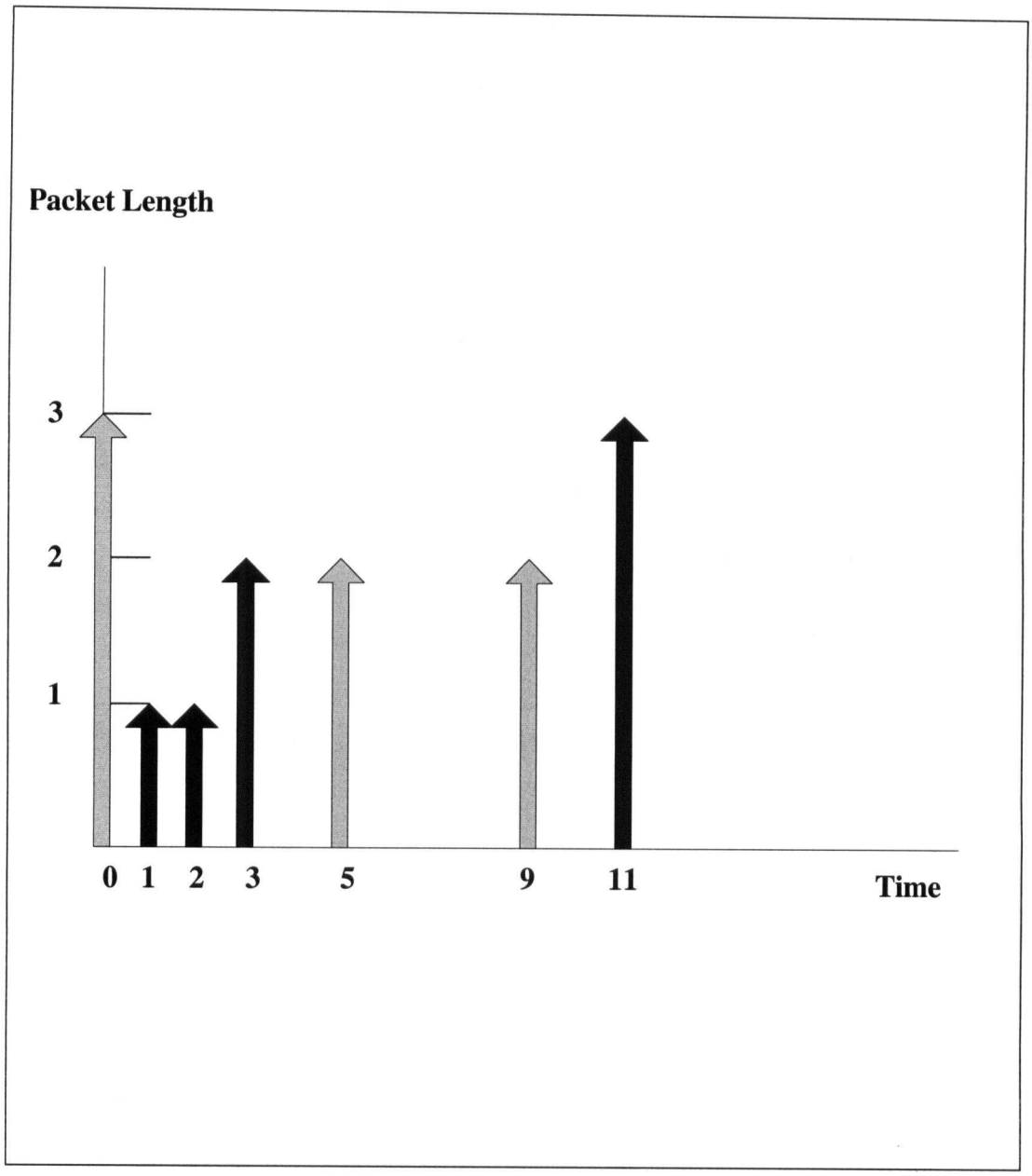

Figure 4.5: Arrivals for both sessions.

	Session 1				*Session 2*		
Arrival	1	2	3	11	0	5	9
Size	1	1	2	2	3	2	2
GPS	3	5	9	13	5	9	11
WFQ	4	5	7	13	3	9	11

Table 4.1: True departure times under GPS and WFQ.

	Session 1				*Session 2*		
Arrival	1	2	3	11	0	5	9
Size	1	1	2	2	3	2	2
GPS	3.11	5	8.894	13	4.89	9	11
WFQ	4	5	7	13	3	9	11

Table 4.2: Departure times when system clock = $111e^{-3}$, time resolution = 0 and timestamp resolution = 0.

as in Figure 4.5. The departure times according to both GPS and WFQ are shown in Table 4.1. To show the effect of time and timestamp resolutions on the simulation output, we set the scheduling clock to a value that is not a divisor of any of the arrival times. This means that the actual arrivals to the system are within a scheduling clock period value of the true arrival times but not really equal to any of them. Table 4.2 shows the resulting departure times.

Note that in Table 4.2 we seem to no longer have any simultaneous departures in GPS. This is due to the fact that the arrivals have been shifted a little to reflect

	Session 1				*Session 2*		
Arrival	1	2	3	11	0	5	9
Size	1	1	2	2	3	2	2
GPS	2.999	4.999	8.999	13	5.001	9	11
WFQ	4	5	7	13	3	9	11

Table 4.3: Departure times when system clock = $111e^{-3}$ and time resolution = $20e^{-3}$ and timestamp resolution = 0.

	Session 1				*Session 2*		
Arrival	1	2	3	11	0	5	9
Size	1	1	2	2	3	2	2
GPS	2.999	4.999	8.999	12.999	4.999	8.999	10.999
WFQ	4	5	7	13	3	9	11

Table 4.4: Departure times when system clock = $111e^{-3}$ and time resolution = $20e^{-3}$ and timestamp resolution = $2e^{-3}$.

	Session 1				*Session 2*		
Arrival	1	2	3	11	0	5	9
Size	1	1	2	2	3	2	2
GPS	3	5	9*	13	5	9**	11
WFQ	4	5	7	13	3	9	11

Table 4.5: Departure times when system clock = $200e^{-3}$ and time resolution = 0 and timestamp resolution = 0.

the scheduling clock value. However, this has no impact on WFQ departure times, as can be seen from the table. To make the GPS simulation more accurate we can increase the time and timestamp resolution. By choosing a time resolution = $20e^{-3}$, we obtain the results in Table 4.3. We notice from Table 4.3 that the accuracy of GPS departures improved a lot by choosing a nonzero value for the time resolution. Also, the WFQ departure times are still correct. However, we still have the case of very close departure times being considered distinct. What we need to do is to modify the timestamp resolution to enable such close departure times to be "lumped together". By choosing a timestamp accuracy = $2e^{-3}$, we obtain the departure times of Table 4.4. We cannot achieve any better GPS simulation results due to the problem of arrival times being non-multiple values of the scheduling clock. However, this has no effect on the departure times of the WFQ scheduler. What

we seek to improve is our GPS simulation. One way of achieving a better GPS simulation is to choose the scheduling clock to be a divisor of all of the arrival times. Table 4.5 shows the departure times when the scheduling clock is equal to $200e^{-3}$.

In Table 4.5, we have shown two different GPS departure times 9* and 9** to indicate that they are both very close to the value 9 but are not equal. Again the WFQ departure times are correct despite the GPS simulation inaccuracy. To solve the problem of close departure times, we increase the value of timestamp resolution to $1e^{-6}$. The results are exactly equal to those in Table 4.1. Note that in all cases the inaccuracies in the GPS simulations do not affect the WFQ scheduler results.

Chapter 5

Experimental Results of the Minimum-Work Per-Flow Packet Scheduler

5.1 Introduction

This chapter provides a summary of results obtained from validation and performance tests for a base-line per-flow packet scheduler currently implemented as a software module in Linux OS. When used with traffic shapers, the scheduler provides precise bandwidth and delay quality-of-service (QoS) guarantees to each flow independently. The scheduler uses the Minimum-Work Weighted Fair Queuing (MW-WFQ) technique described in this thesis, which offers a scalable and significantly

faster implementation of the well-established weighted fair-queuing algorithm.

Test Highlights:

1. Comparison to Other Schedulers: Test results show that MW-WFQ provides the required QoS guarantees for each shaped flow. The other two schedulers tested (FIFO, and Self-Clocked Fair Queuing) fail to provide similar guarantees, even when the number of flows is small and each flow is strictly shaped by a token-bucket traffic regulator.

2. Performance Profiling and Scalability: Profiling results show that MW-WFQ has a fixed and very short per-packet computation interval. This shows that the maximum throughput (packets-per-second) achieved by MW-WFQ over a link is independent of the number of flows. Profiling also shows that typical implementations of ideal WFQ suffer from variable processing delays proportional to the number of flows in the scheduler. For the cases considered in this profiling, MW-WFQ per-packet processing can be as much as 90 folds faster than a typical implementation of ideal WFQ.

3. Time-Stamp Correctness: Test results show that the MW-WFQ scheduler produces timestamps identical to those produced by the ideal WFQ algorithm for all packets from all participating sessions. Stated differently, MW-WFQ dispatches packets from different sessions in precisely the same order as the ideal WFQ scheduler.

5.2 Delay Guarantees

5.2.1 Test setup

The test setup is comprised of a real-time load generator and tester feeding packets to PC-based software Linux router. The load generator and tester is the Adtech AX/4000 from Spirent Communications. The PC rack module uses an Intel 533MHz Celeron chip running Linux OS, version 2.4.2.1 (included in Redhat Linux 7.1). The specifications of the equipment and software are given in Table 5.1.

The test bed configuration is shown in Figure 5.1. The connection between the Adtech AX/4000 tester and the PC-based Linux router is asymmetric to emulate the effect of converging traffic at the output line card of the router. The AX/4000 sends packets to the Linux router using a 100Base-T Fast Ethernet connection (100 Mbps), while the Linux router sends scheduled packets back to the Adtech tester through 10Base-T Ethernet connection (10 Mbps). This emulates the effect of having the equivalent of up to ten 10Base-T Ethernet ports concurrently sending packets to the output port.

The traffic profile was set up such that total traffic reaching the Linux router has a long-term average rate of 10 Mbps. During a traffic burst, however, all the packets in the burst are transmitted back-to-back to the router at the full 100 Mbps rate. The MW-WFQ packet scheduler is implemented in the output line-card that forwards packets at a maximum rate of 10 Mbps rate.

Equipment / Software	Description / Usage
Rack Module (Soft Router)	Intel ISP1100 with a 533 MHz Celeron processor, 440BX chipset, PC100 RAM, and 82559 Ethernet controllers
Line Cards	100Base-TX and 10Base-TX
Linux Software	Linux OS, version 2.4.2.1 included in Redhat Linux 7.1
Packet Scheduler	MW-WFQ scheduler embedded as a Linux kernel module
Load Generator / Tester	Adtech AX/4000 from Spirent Communications

Table 5.1: System specifications.

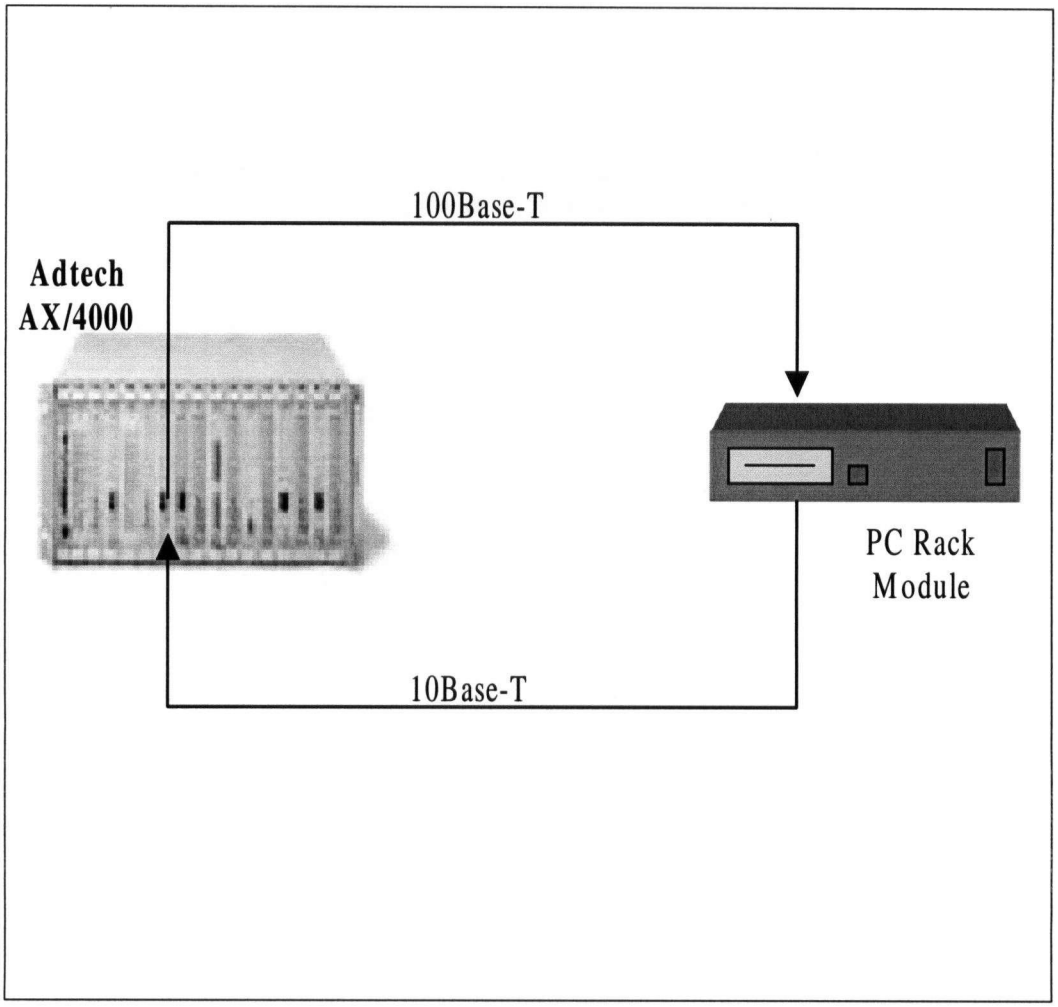

Figure 5.1: Test setup.

The Adtech AX/4000 was used to generate, multiplex, then send packets from 3 flows each having its own token-bucket shape to the Linux router through a 100Base-TX line. The Adtech AX/4000 device also captures the scheduled packets that are sent out of the Linux router through the 10Base-T line. For additional verification and delay analysis, the arrival time and departure time computed by the scheduler for each packet are dumped into a log file.

For verification and performance comparison, four types of schedulers were implemented in the Linux Kernel, and the tests were repeated with the same traffic scenarios for each scheduler. The schedulers are:

1. MW-WFQ (Minimum-Work Weighted Fair Queuing).

2. WFQ (Weighted Fair Queuing): Full implementation of the ideal WFQ scheduler. The time stamps it produced are compared against MW-WFQ to ensure full conformance.

3. SCFQ (Self-Clocked Fair Queuing): An approximation of WFQ that uses highly simplified time-stamp calculations to speed up processing time.

4. FIFO (First-In First-Out): a very common and very simple scheduling method. Used also with multi-priority schedulers to service the packets within each class.

Remark: We assume that all three flows belong to the same highest priority QoS class on the output link.

5.2.2 Results

Table 5.2 provides a description of the traffic sources used in the test. The constant bit-rate (CBR) source for flow 1 represents a typical voice-over-IP flow, mixed with other higher-rate flows such as a video stream (flow 2), and an "aggregated" flow with a much higher rate (flow 3).

Table 5.3, compares the maximum queuing delays achieved by the MW-WFQ scheduler against the theoretical delay bounds in equation 2.6, obtained by applying the ideal WFQ to each of the flows. The table also shows the average packet delay. It is clear that MW-WFQ achieves precisely the theoretical delay bounds for all flows. Tables 5.4 and 5.5 show similar comparisons for the FIFO and SCFQ schedulers, respectively.

Table 5.4 clearly shows that even with such a small number of flows, FIFO queuing violates the delay bounds for two of the three flows. The delay bound violation is particularly large for flow 1, the low-bandwidth CBR flow. SCFQ also violates the delay bound for flow 1 as evident from Table 5.5. This result is significant in that it shows that approximations of WFQ, such as SCFQ, can fail to provide delay guarantees even with a very small number of flows.

Table 5.6 compares all three schedulers against the theoretical delay bounds, and Table 5.7 compares FIFO with MW-WFQ in terms of inter-packet delay variation (or jitter). Maintaining a small inter-packet jitter is important for real-time continuous

Flow number	*Flow Description*	*Avg. Rate) (kbps)*	*Packet Size (Bytes)*	*Max. Burst) (Packets)*
1	Constant Bit-Rate	32	80	1
2	Bursty	968	1000	50
3	Bursty	9000 (9 Mbps)	1500	100

Table 5.2: Traffic Sources (Token-Bucket Shaped).

Flow number	*Avg. Rate) (kbps)*	*Max. Theoretical Delay (ms)*	*Max. Measured Delay (ms)*	*Average Delay (ms)*
1	32	41.2	41.7	28.5
2	968	414.7	280.7	78.4
3	9000 (9 Mbps)	134.9	132.4	69.7

Table 5.3: MW-WFQ: Measured Packet Delays over 10 Mbps Link.

Flow number	*Avg. Rate) (kbps)*	*Max. Theoretical Delay (ms)*	*Max. Measured Delay (ms)*	*Average Delay (ms)*
1	32	41.2	142.9	65.8
2	968	414.7	53.7	22.0
3	9000 (9 Mbps)	134.9	145.6	75.7

Table 5.4: FIFO Queuing: Measured Packet Delays over 10 Mbps Link.

Flow number	*Avg. Rate) (kbps)*	*Max. Theoretical Delay (ms)*	*Max. Measured Delay (ms)*	*Average Delay (ms)*
1	32	41.2	109.6	58.0
2	968	414.7	280.7	83.3
3	9000 (9 Mbps)	134.9	132.6	69.0

Table 5.5: SCFQ: Measured Packet Delays over 10 Mbps Link.

Flow number	*Max. Theoretical Delay (ms)*	*MW-WFQ Delay (ms)*	*FIFO Delay (ms)*	*SCFQ Delay (ms)*
1	41.2	41.7	142.9	109.6
2	414.7	280.7	53.7	280.7
3	134.9	132.4	145.6	132.6

Table 5.6: Maximum Measured Packet Delay over 10 Mbps Link.

Flow number	*Avg. Rate (kbps)*	*Max. Theoretical Delay (ms)*	*MW-WFQ Max. Jitter (ms)*	*FIFO Max. Jitter (ms)*
1	32	41.2	61.6	162.8
2	968	414.7	259.1	378.9
3	9000 (9 Mbps)	134.9	28.2	41.3

Table 5.7: Maximum Measured Packet Delay-Jitter (At receiver, 10 Mbps Link).

multimedia traffic.

5.3 Scheduler Speed

5.3.1 Performance Profiling Test

To show the effect of equal timestamps on the performance of WFQ implementation, we profiled a standard implementation on a RISC processor. The goal of performance profiling test is to measure the maximum speed of the MW-WFQ scheduler on a specific processor platform. The scheduler speed, measured in packets-per-second, gives an indication of the number of flows that can be handled simultaneously by the scheduler. We have chosen to profile the scheduler speed on a simulator of a 200 MHz StrongArm SA-110 RISC processor, a core engine for a number of network processors. The scheduler speed is determined by measuring the maximum number of processor cycles used to calculate the timestamp of a packet, plus the time it takes to place the packet in the correct position in the packet queue. Note that the time a packet spends in the packet queue before being sent on the output link is part of the queuing delay and therefore is not included in the processing delay.

Precise WFQ computations are normally slowed down considerably by timestamp computations for packets arriving during busy (backlog) system periods. The processing delay problem is caused by accumulation of simultaneous departures in the ideal "fluid" GPS scheduler. Ideal WFQ schedulers simulate GPS to obtain their timestamps.

5.3.2 Results

Table 5.8 shows the number of cycles required to compute the time for a new packet arrival, assuming that 1, 10, 50, and 200 simultaneous departures can occur in the fluid GPS scheduler. It also reports the speedup achieved by the MW-WFQ scheduler over standard implementation of the ideal WFQ algorithm. Two types of cycles are compared for each GPS-departure category: instructions and core cycles. The advantage of MW-WFQ increases as the probability of more GPS departures increases. While MW-WFQ processing remains essentially constant, the processing time of ideal WFQ increases linearly with the number of simultaneous GPS departures. For example, with 200 simultaneous departures, WFQ requires 129190 core cycles to compute the timestamp for a single packet arrival, while MW-WFQ computes the same timestamp in 1414 core cycles achieving a speedup of 91 over the standard WFQ.

Simultaneous Departures	*Cycle Type*	*MW-WFQ (No. of Cycles)*	*Standard WFQ (No. of cycles)*	*Speedup (MW-WFQ)*
200	Instructions	619	85157	138
	Core Cycles	1414	129190	91
50	Instructions	627	21907	35
	Core Cycles	1424	33591	26
1	Instructions	626	673	1.07
	Core Cycles	1460	1668	1.14

Table 5.8: WFQ Per-Packet Processing Delay As Function Of Processor Cycles (STRONGARM SA-110 RISC Processor - 200Mhz).

5.4 Timestamp Validation

This section presents the method used to verify that the MW-WFQ scheduler assigns timestamps to packets from participating sessions in full accordance with the ideal WFQ algorithm. The purpose is to show that MW-WFQ orders packet departures in the same exact order as the ideal WFQ while using a much shorter computation time than any standard implementation of WFQ.

Traffic input: Three shaped flows having a total average bit rate equal to 10Mbps are applied as input as follows:

1. Flow 0: 32kbps, 80 Bytes/packet, constant bit rate.

2. Flow 1: 968kbps, 1000 Bytes/packet, maximum burst size is 50 packets.

3. Flow 2: 9Mbps, 1500 Bytes/packet, maximum burst size is 100 packets.

Traffic output: Packets are scheduled according to their timestamps by transmitting the packet with the smallest timestamp first.

Validation procedure: Using the known arrival times of all the packets in the test, we can compute the timestamps of each packet as follows:

1. If the arrival is to an idle flow k in the GPS system, then it receives a timestamp of:

$$TS = V(t_a) + \frac{L}{r_g^k}$$

Where t_a is the packet's arrival time, $V(t_a)$ is the virtual time at time of arrival of packet, L is the packet length and r_g^k is the flow's guaranteed rate (in this test it equals to the average rate of the kth flow).

$V(t)$ is a piecewise linear function of time that starts from a value of zero at time zero and has a slope S equal to the ratio of total link bandwidth C (10Mbps) to the sum of guaranteed rates of all active flows at time t, as follows:

$$S = \frac{C}{\sum_{j \in A(t)} r_g^j}$$

Where $A(t)$ is the set of active flows at time t.

2. If the arrival is to an active flow in GPS, then the timestamp is:

$$TS = TS_{previous} + \frac{L}{r_g}$$

where $TS_{previous}$ is the timestamp of the previous packet to arrive to that flow.

3. The GPS system services all active flows simultaneously such that active flow k receives an instantaneous bit rate of:

$$r(t) = \frac{r_g^k C}{\sum_{j \in A(t)} r_g^j}$$

The above formulas were used to verify that the timestamps, produced by the software implementing MW-WFQ, are identical to the timestamps resulting from applying the ideal WFQ scheduler. The results obtained from applying the verification procedure on a trace of over 16,000 packets from the three flows listed above show that the calculated and measured timestamps agreed perfectly for each packet from every flow.

Chapter 6

Conclusion and Future Work

The importance of per-flow schedulers lies in the fact that they provide strict delay and fairness guarantees. In particular, it is well-known that WFQ has the best characteristics among all the well know per-flow scheduling techniques. However, for some time now, it has been accepted, without proof, that WFQ requires $O(N)$ computational complexity where N is the number of sessions sharing a link. When N is large, such as the case in a metropolitan or wide-area network, the computational cost becomes too high. This thesis showed that the cause of this high computational complexity is not the "iterated-deletion" as was commonly accepted, but rather the simultaneous departures that may take place in GPS. A new algorithm that solves this problem of $O(N)$ complexity was presented. This enabled the implementation of WFQ as a real-time traffic scheduler on a Linux box. Test results were presented to show that the new algorithm enables WFQ to meet all delay requirements of the

scheduled flows.

Future work related to this thesis include the following:

- Finding a meaningful translation between the QoS requirements of applications and those of WFQ. This enables Service Level Agreements (SLA's) to be translated into token-bucket parameters.

- Investigating the effect of dynamic shares on the delay and fairness properties of a Weighted Fair Queuing (WFQ) Scheduler. This helps us to deal with sessions that have time-varying token-bucket parameters. In addition, we expect the set of active sessions to change over time as new sessions are created and old ones terminate.

- Studying the effect of aggregation on WFQ

- Proposing a new Label Distribution Protocol in MPLS, based on WFQ rather than Diffserv.

- Studying the effects on the end-to-end guarantees of flows when certain nodes along the path of these flows are not QoS-aware

Bibliography

[1] D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. Technical Report RFC 2475, IETF, Dec. 1998.

[2] Anna Charny and J. Y. Le Boudec. Delay bounds in a network with aggregate scheduling. *Proc. QOFIS*, October 2000.

[3] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. Technical Report RFC 3031, IETF, Jan. 2001.

[4] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The singlenode case. *IEEE/ACM Trans. Networking*, 1(3):344–357, June 1993.

[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 Trans. Networking*, 2(2):137–150, April 1994.

[6] A. Demers, S. Keshav, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. *Internetworking Res. and Experience*, 1, 1990.

[7] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control- The Single Node Case. *Proc. IEEE INFOCOM '92*, 2:915–24, May 1992.

[8] D. Stiliadis and A. Varma. Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms. *Proc. IEEE INFOCOM '96*, pages 111–19, April 1996.

[9] D. Stiliadis and A. Varma. Rate-Proportional Servers: A Design Methodology for Fair Queuing Algorithms. *IEEE/ACM Trans. Networking*, 6(2):164–74, April 1998.

[10] D. Stiliadis and A. Varma. Efficient Fair-Queuing Algorithms for Packet-switched Networks. *IEEE/ACM Trans. Networking*, 6(2):175–85, April 1998.

[11] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. *ACM Trans. Comp. Sys.*, 9:101–24, May 1991.

[12] S. Golestani. A Self-Clocked Fair Queuing Scheme for Broadband Applications. *Proc. IEEE INFOCOM '94*, pages 636–46, April 1994.

[13] J. R. Bennett and H. Zhang. Hierarchical Packet Fair Queuing Algorithms. *Proc. ACM SIGCOMM '96*, pages 143–56, Sept. 1996.

[14] F. M. Chiussi and A. Francini. Minimum-Delay Self Clock Fair Queueing Algorithm for Packet-Switched Networks. *Proc. IEEE INFOCOM '98*, pages 1112–21, March 1998.

[15] F. M. Chiussi and A. Francini. Implementing Fair Queueing in ATM Switches: The Discrete-Rate Approach. *Proc. IEEE INFOCOM '98*, pages 272–81, March 1998.

[16] P. Goyal, H. M. Vin, and H. Haichen. Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. *IEEE/ACM Trans. Networking*, 5(5):690–704, Oct. 1997.

[17] R. Brown. Calendar Queues: A Fast O(1) Priority Queue Implementaion for the Simulation Event Set Problem. *Communications of the ACM*, 31(10):1220–27, Oct. 1988.

[18] S. Keshav. *An Engineering Approach to Computer Networking.* AddisonWesley Publishing Company, 1997.

[19] H. Tayyar and H. Alnuweiri. Weighted Fair Queuing Scheduler. world-wide patent WO0143347A, June 2001. http://wo.espacenet.com.

[20] M. Kazemi-Nia and H. Alnuweiri. A Systolic Parallel Priority Queue (PPQ)with Output Rate-Control for High Speed Networks. submitted for publication in the IEEE Transactions on VLSI Systems, June 1999.

[21] J. R. Bennett and H. Zhang. WF2Q: Worstcase fairweighted fair queueing. *Proc. IEEE INFOCOM96*, pages 120–128, March 1996.

[22] J.Wroklawski. The Use of RSVP with IETF Integrated Services. Technical Report RFC 2210, IETF, Sept. 1997.

[23] J.Wroklawski. Specification of the Controlled-Load Network Element Service. Technical Report RFC 2211, IETF, Sept. 1997.

[24] S.Shenker, C.Partridge, and R.Guerin. Specification of Guaranteed Quality of Service. Technical Report RFC 2212, IETF, Sept. 1997.

[25] R. Cruz. Service burstiness and dynamic burstiness measures: A framework. *Journal of High Speed Networks*, 1(2):105–127, 1992.

[26] R. Cruz. Quality of service guaranteed in virtual circuit switched network. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, August 1995.

[27] L. Georgiadis, R. Guerin, , and V. Peris. Efficient network QoS provisioning based on per node traffic shaping. *Proc. IEEE INFOCOM96*, March 1996.

[28] P. Goyal and H. M.Vin. Fair airport scheduling algorithms. *Proc. NOSSDAV97*, May 1997.

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

[30] H. Sariowan, R.L. Cruz, and G.C. Polyzos. Scheduling for quality of service guarantees via service curves. *Proc. Intl. Conf. on Computer Communications and Networks (ICCCN)*, pages 512–520, September 1995.

[31] I. Stoica and H. AbdelWahab. Earliest eligible virtual deadline first: A flexible and accuratemechanismfor proportional share resource allocation. Technical Report TR9522, Old Dominion University, November 1995.

[32] I. Stoica, H. AbdelWahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton. Aproportional share resource allocation for realtime, timeshared systems. *Proc. IEEE RTSS'96*, pages 288–289, December 1996.

[33] I. Stoica, H. Zhang, and T.S.E. Ng. A hierarchical fair service curve algorithm for link sharing, realtime and priority services. Technical Report CMUCS97154, Carnegie Mellon University, July 1997.

[34] Z.Liu, Z.L. Zhang, and D. Towsley. Closed form deterministic performance bounds for the generalized processor sharing scheduling discipline. To appear journal of Combinatorial Optimaization, 1997.

[35] H. Zhang. Service disciplines for guaranteed performance service in packet switching networks. *Proc. IEEE*, 83(10):1374–1399, October 1995.

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

[37] J. Bennett and H. Zhang. Worst-case Fair Weighted Fair Queuing. *Proc. INFOCOM'95*, 1995.

[38] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[39] N. Figuera and J. Pasquale. Leave-in-time: A new service discipline for real-time communication in a packet switching data network. *Proc. Sigcomm 95*, September 1995.

[40] P. Goyal, S. S. Lam, , and H. M. Vin. Determining End-to-End Delay Bounds in Heterogeneous Networks. *Proc. Workshop on Network and OS Support for Audio- Video*, pages 287–298, April 1995.

[41] D. Stiliadis and A. Varma. A general methodology for designing efficient traffic scheduling and shaping algorithms. *Proc. IEEE INFOCOM97*, pages 326–335, April 1997.

[42] B. H. Choi and H. S. Park. Rate Proportional SCFQ Algorithm for High-Speed Packet-Switched Networks. *ETRI Journal*, 22(3), Sept. 2000.

[43] J. C. R. Bennett, K. Benson, A. Charny, W. F. Courtney, and J.-Y. Le Boudec. Delay jitter bounds and packet scale rate guarantee for expedited forwarding. In *ACM/IEEE Transactions on Networking*, 2002.

[44] P. Mannersalo and I. Norros. GPS schedulers and Gaussian traffic. In *Proc. Infocom*, 2002.

[45] N. Christin and J. Liebeherr. The QoSbox: A PC-router for quantitative service differentiation in IP networks. Technical Report CS-2001-28, University of Virginia, November 2001. ftp://ftp.cs.virginia.edu/pub/techreports/CS-2001-28.pdf. In submission.

[46] A. Stamoulis and G. B. Giannakis. Deterministic Time-Varying Packet Fair Queueing for Integrated Services Networks. *VLSI Signal Processing*, 2002.

[47] Nan Ni Laxmi. Fair Scheduling and Buffer Management in Internet Routers. *INFOCOM 2002.*

[48] Robert Denda, Albert Banchs, and Wolfgang Effelsberg. The Fairness Challenge in Computer Networks. In *QofIS*, pages 208–220, 2000.