ABSTRACT

AMUDALA BHASKER, AJAY BABU. Tiered-Service Fair Queueing (TSFQ): A Practical and Efficient Fair Queueing Algorithm. (Under the direction of Professor George N. Rouskas.)

A router in today's Internet has to satisfy two important properties in order to efficiently provide the Quality of Service (QoS) requested by the users. It should be fair among flows and also have low operational complexity. The packet scheduling techniques that have been proposed earlier do not have both these properties. Schedulers like Weighted Fair Queueing (WFQ) provide good fairness among flows but have high operational complexity. Schedulers like Weighted Round Robin (WRR) are efficient but provide poor fairness among flows. We propose a new packet scheduling technique, Tiered Service Fair Queueing (TSFQ), which is both fair and efficient. We achieve our goal by applying the concept of traffic quantization. A quantized network offers a small set of service levels (tiers), each with its own weight. Each flow is then mapped to one of the service levels so as to guarantee a QoS at least as good as that requested by the flow. We propose different versions of TSFQ, each with its own level of fairness. We present the complexity analysis of the TSFQ scheduler. Finally, we demonstrate through simulations on the TSFQ implementation on $ns-2$, that TSFQ provides good fairness among flows.

Tiered-Service Fair Queueing (TSFQ): A Practical and Efficient Fair Queueing Algorithm

by

AJAY BABU AMUDALA BHASKER

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2006

Approved By:

Biography

Ajay Babu Amudala Bhasker was born in Tirupathi and brought up in Chennai, India. After finishing his high school in Chennai, he graduated with a Bachelor of Technology (B.Tech) degree in Information Technology from Sri Venkateswara College of Engineering, University of Madras, Chennai. He joined the department of Computer Science at the North Carolina State University, Raleigh, NC in fall of 2004. He is currently working towards completion of his master's degree in Computer Science.

Acknowledgements

I would like to express my most sincere gratitude to Professor George Rouskas for having guided me at every step of this research work. This thesis would not have been possible without his constant support and advice. I am deeply indebted to him for his patience and invaluable suggestions during the course of this thesis.

I am also thankful to Dr. Rudra Dutta and Dr. Khaled Harfoush for serving on my thesis committee. I would like to acknowledge the National Science Foundation for supporting this research.

I would like to thank Zyad Dwekat for all his suggestions throughout this work. I am grateful to all my friends for their help and support.

Above all, I am grateful to my parents for their love. I am also grateful to my brother, Vinodh, for having stood by me in every decision I have made.

Contents

List of Figures

Chapter 1

Introduction

1.1 Packet Scheduling

Technological advances have dramatically increased electronic processing speeds, however so has the transmission capacity of the network links. With the advances in the optical network technologies, the data rate of the network links has increased to 10 Gbps, currently and soon to 40 Gbps. The exponential increase in the amount of bandwidth available within the network has outpaced the improvements in switching and routing capabilities, and this is expected to continue in the future. So, there is a severe mismatch between the bandwidth supported by the network links and the operational speed of the electronic switching and/or routing functions at the network nodes. One such important function of the routing equipment is packet scheduling or queueing.

Packet scheduling refers to the decision process used to select the order in which the packets are transmitted onto the outgoing link. Packet schedulers can be classified into two types:

Work conserving schedulers In these schedulers, the link is never idle when there are packets waiting for service. Most of the existing packet schedulers belong to this type.

Non-work conserving schedulers In these schedulers, the link may be idle even if it has packets to serve. One important reason for delaying the service of packets is to reduce the delay jitter.

Packet scheduling is an important means of controlling congestion and providing specific Quality of Service (QoS) to the flows in the networks. Determining the order in which the packets are serviced is a challenging task because of the QoS requirements of the flows. Each flow in applications such as the World Wide Web (WWW) that involve transfers of data, voice clips and video images have tight bounds on several performance measures such as delay, loss rate, delay jitter, etc. There is also an exponential increase in the number of such flows, making it imperative for the packet scheduler to be very efficient in terms of operational time and also at the same time serve the flows according to their specific QoS requirements.

The goal of this research is to design a practical and efficient packet scheduling scheme. The three important properties of an ideal packet scheduler is as follows:

- Since the schedulers are used in high speed networks, they should have a low operational time complexity, preferably O(1).
- The schedulers should maintain delay bounds for guaranteed-service applications.
- The scheduler must provide fairness among the flows competing for the shared link. In other words, the scheduler must provide fair sharing of bandwidth among the flows. Short-term throughput is a good measure for determining the fairness among the flows.

Few packet scheduling schemes like Weighted Fair Queueing (WFQ) provide good fairness among flows and QoS guarantees, but have relatively high complexity O(log n), where n is the number of flows in the system. Other packet scheduling techniques like Weighted Round Robin (WRR) have an O(1) complexity, but in general do not have good fairness and bounded delay properties.

1.2 Organization of Thesis

The thesis is organized as follows. In Chapter 2, we describe traffic quantization and its applications in packet-switched networks. We then, look at the approaches to solve the problem of quantization. In Chapter 3, we describe Generalized Processor Sharing (GPS), the ideal packet scheduling technique, and two of its emulations namely Weighted Round Robin (WRR) and Weighted Fair Queueing (WFQ). In Chapter 4, we propose a new packet scheduling technique, Tiered Service Fair Queueing (TSFQ), with different versions that apply traffic quantization to achieve better efficiency and fairness. In Chapter 5, we present the architecture and implementation details of the TSFQ component we implemented on $ns - 2$ simulator and present some fairness-related performance results in Chapter 6. Finally we summarize our work in Chapter 7 and provide directions for future research.

Chapter 2

Traffic Quantization: Applications and Algorithms

In this chapter, we look at the need for applying quantization in packet-switched networks. Also, we look at the p-median problem and the approach to solve the problem of quantization by mapping it to an instance of the p-median problem. We look into the directional p-median problem, a special case of the traditional p-median problem, and the solutions to this problem [9]. We then look at an extension to the directional p-median problem and its solution [3]. Finally, we look at the findings made by comparing the quantized networks with the continuous networks [9].

2.1 Traffic Quantization and its Applications

Traffic quantization can be defined as the process of mapping each flow in the network to one of a small set of service levels (tiers), in such a way that a Quality of Service (QoS) at least as good as that requested by the flow is guaranteed. Traffic quantization trades off a small amount of system resources for simplicity in the core network functions, including packet scheduling, traffic policing, etc.

Due to the advances in optical transmission technologies, there is an exponential increase in the amount of bandwidth available within a network. Because of this, a router in today's high speed networks serves hundreds of thousands of flows, each with different QoS requirements. But the ability of routing functions at the network nodes to support per-flow functionality in continuous networks is limited. In a quantized network, per-flow functionality could be supported in an efficient and scalable manner. A quantized network offers a small set of service levels (tiers) and each flow in the network is mapped to one of the tiers, in such a way that a QoS at least as good as requested by the flow is guaranteed. This simplifies a wide range of network functions. For example, consider the issue of traffic policing. In a continuous-rate network, one flow may request a bandwidth of 99.92 Kbps while another 99.98 Kbps. In this case, the network provider faces a difficult task of distinguishing these two rates and enforcing them reliably. A quantized network might assign both the flows to the next higher level of bandwidth, say 100 Kbps. Now, there is no need for the network provider to distinguish between these two flows. The network operator only needs to supply policing mechanisms for a small set of rates, independent of the number of flows. Traffic quantization may also simplify other network functions such as packet scheduling, traffic engineering, state dissemination, network management, service level agreements, billing, etc.

The disadvantage of a quantized network is that it may consume more resources than a continuous-rate network to satisfy the same set of requests for service. In the case of bandwidth quantization, the total amount of bandwidth required, to service the same set of requests, may be more in a quantized network than in a continuous-rate network. In order to minimize the additional amount of resources, it is important to select an optimal set of service levels. The algorithms presented in the following sections help in determining an optimal set of service levels, given a set of service requests.

2.2 The p-Median Problem

The p-median problem [8] belongs to the class of location problems, where the objective is to select a subset of n points, referred to as the set of demand points, as the location of setup facilities, called the supply points, that will serve the demand points. The utility of the demand point is defined as a function of the distance between it and the closest supply point that services it. The objective here is to minimize the overall utility of the

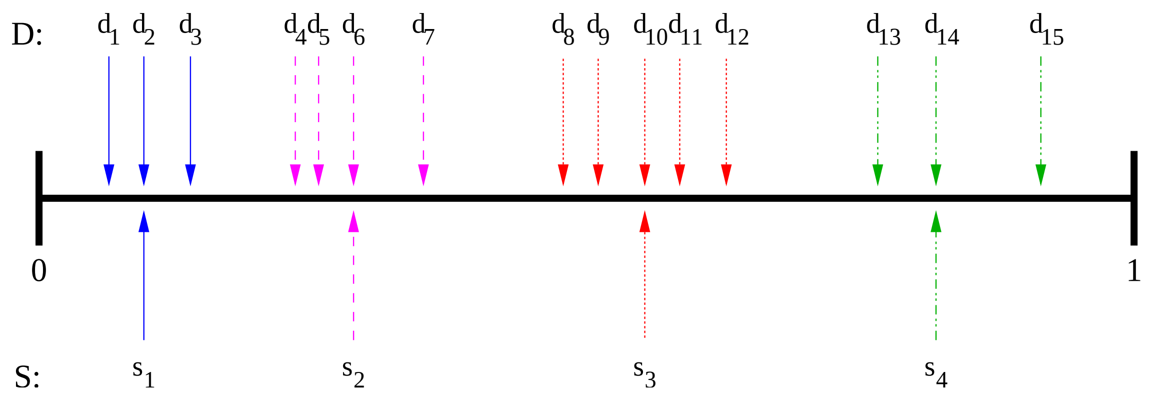

Figure 2.1: Mapping a set of demand points D to a set of supply points S in traditional p-median problem on the real line

system by choosing appropriate supply points.

The objective of the p-median problem is to map a set of n demand points $D = \{d_1, d_2, \ldots, d_n\}$ to a set of p supply points $S = \{s_1, s_2, \ldots, s_p\}$ in such a way that the total distance between each demand point and its closest supply point is minimum. In this problem, p is always less than or equal to n and the set of p supply points chosen should be a subset of the n demand points.

The p-median problem, as a location problem, is usually defined for points on the plane, but can be generalized to points in a k-dimensional system, $k \geq 1$. In this work, we will focus on the p-median problem when $k = 1$, which we will refer to as p-median problem on the real line. Figure 2.1 shows an instance of the p-median problem on the real line where $\mid D \mid = n = 15 \ and \ \mid S \mid = p = 4$. Down arrows represent demand points, and up arrows represent supply points. This instance is such that demand points $d_1 - d_3$ are mapped to supply point s_1, demand points $d_4 - d_7$ are mapped to supply point s_2, demand points $d_8 - d_{12}$ are mapped to supply point s_3, and demand points $d_{13} - d_{15}$ are mapped to supply point s_4.

Formally, the p-median problem on the real line can be formulated as

$$Minimize \sum_{i=1}^{n} \bar{\omega}(d_i, S) \tag{2.1}$$

subject to

$$d_1 \leq d_2 \leq \ldots \leq d_N \tag{2.2}$$

$$\bar{\omega}(d_i, S) = \min\{\mid d - d_i \mid : d \in S\} \tag{2.3}$$

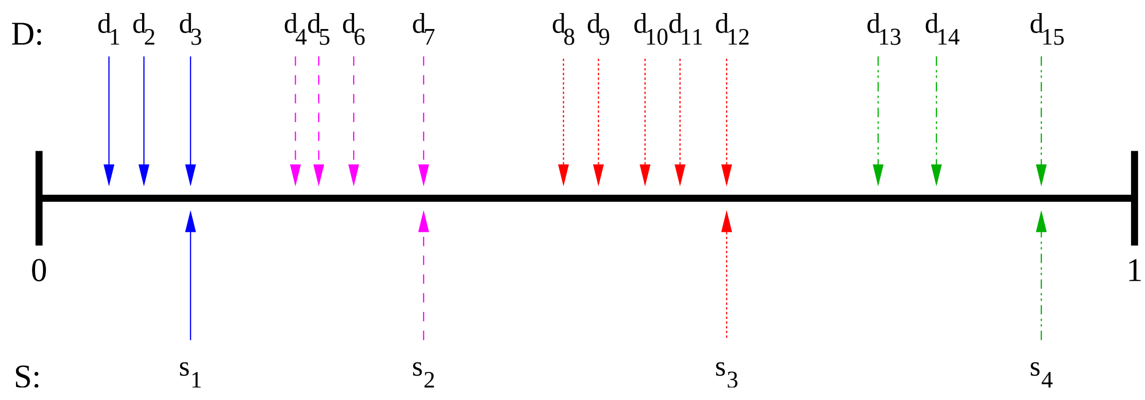

Figure 2.2: Mapping a set of of demand points D to a set of supply points S in directional p-median problem on the real line

$$S \subseteq D \tag{2.4}$$

$$|S| = p \tag{2.5}$$

$$d_i \in D \tag{2.6}$$

$$1 \leq p \leq n \tag{2.7}$$

where $\bar{\omega}(d_i, S)$ is the distance between d_i and its nearest supply point.

2.3 The Directional p-Median Problem

The directional p-median problem [9] is a special case of the traditional p-median problem. The directional p-median problem on the real line has an additional restriction that the nearest supply point for a given demand point has to be greater than or equal to the demand point itself. Figure 2.2 presents an instance of the directional p-median problem on the real line. The demand set D is identical to that in Figure 2.1, and the number of supply points $p = 4$ as well. However, as we can see, the additional constraint of the directional problem leads to a different solution. For instance, demand points $d_1 - d_3$ are mapped to the supply point s_1, but unlike in Figure 2.1, we have that all the demand points are less than or equal to s_1 as required by the problem definition. Note also, that for a solution to be feasible, the largest supply point (in this case, s_4) must be greater than or equal to the largest demand point (in this case, d_{15}). Formally, the directional p-median problem on the real line is: Given a set $D = \{d_1, d_2, \ldots, d_n\}$ of n demand points that are sorted in non-decreasing order, find an optimal set $S = \{s_1, s_2, \ldots, s_p\}$ of p supply points, which minimizes the following objective function:

$$Obj(S) = \sum_{j=1}^{p} \sum_{d_i \in D_j} (s_j - d_i) \tag{2.8}$$

subject to

$$S \subseteq D \tag{2.9}$$

$$|S| = p \tag{2.10}$$

$$d_1 \leq d_2 \leq \ldots \leq d_N \tag{2.11}$$

$$s_1 \leq s_2 \leq \ldots \leq s_P \tag{2.12}$$

$$s_{j-1} < d_i \leq s_j \tag{2.13}$$

$$D_j \subseteq D \tag{2.14}$$

Here, D_j denotes the set of demand points that are mapped to the supply point s_j. Since the problem is directional, all elements in the set D_j are greater than s_{j-1} and lesser than or equal to s_j. The sets $D_1, D_2, \ldots, D_p$ are exclusive subsets of D.

An application of the directional p-median problem is the problem of traffic quantization. The QoS requests of the flows and the service levels (tiers) to which they are mapped in a quantized network are analogous to the demand points and supply points, respectively, in the directional p-median problem. The directional constraint in the problem assures that the flows receive QoS at least as good as that requested by them. The objective function $Obj(S)$ corresponds to the excess amount of network resources consumed and the goal of the problem is to minimize this excess resource consumption.

2.3.1 Solutions to the Directional p-Median Problem

A simple approach to solve the directional p-median problem is to use the dynamic programming technique. Given a set $D = \{d_1, d_2, \ldots, d_n\}$ of demand points that are sorted in non-decreasing order, the optimal set $S = \{s_1, s_2, \ldots, s_p\}$ can be determined using the following dynamic programming algorithm, as described in [9]. Let the cost function $\Psi(R_n, p)$ denote the minimum cost for mapping all points in set R_n to p supply points where R_n is the set of n smallest demand points.

$$Obj(S) = \Psi(R_N, p) - \sum_{i=1}^{n} d_i \tag{2.15}$$

$\Psi(R, P)$ can be computed recursively using:

$$\Psi(R_n, 1) = nd_n, n = 1, \ldots, N \tag{2.16}$$

$$\Psi(R_1, p) = d_1, p = 1, \ldots, P \tag{2.17}$$

$$\Psi(R_n, p+1) = \min_{q=l,\ldots,n-1} \{\Psi(R_q, p) + (n-q)d_n\},\ p = 1, \ldots, P-1; n = 2, \ldots, N \tag{2.18}$$

This algorithm has an overall time complexity of $O(n^2p)$.

In order to obtain a better complexity bound, [3] used the property of totally monotone matrices and the Monge property to solve this problem. A matrix $\mathbf{M}$ having real entries is said to be **monotone** if $i_1 > i_2$ implies that $j(i_1) \geq j(i_2)$ where $j(i)$ is the index of the leftmost column containing the maximum value in row i of $\mathbf{M}$. $\mathbf{M}$ is said to be **totally monotone** if its sub matrices are monotone [2]. The directional p-median problem can be represented as a Directed Acyclic Graph (DAG), where each arc weight $\omega(i, k)$ is the sum of the distances between the demand points $d_{i+1}, d_{i+2}, \ldots, d_k$ and the demand point d_k. A complete and weighted DAG satisfies the concave Monge condition if

$$\omega(i, j) + \omega(i+1, j+1) \leq \omega(i, j+1) + \omega(i+1, j) \tag{2.19}$$

for all $0 < i + 1 < j < n$. It has been shown that the DAG representing the directional p-median problem obeys the concave Monge condition [9]. When $n \geq m$, the Monge property can be used to find the minimum entry in each row or column of a totally monotone $n \times m$ matrix in time $O(n)$ [2]. The dynamic programming algorithm in (2.16) - (2.18) was modified in [3] to exploit the Monge condition. A new term $\varpi(i, j)$ was introduced into the recursive equation, which is defined as the cost of mapping demand points $d_{i+1}, d_{i+2}, \ldots, d_k$ to point d_k.

$$\varpi(i, k) = \begin{cases} 0 & \text{if } k = i + 1 \\ (k - i - 1)d_k - \sum_{j=i+1}^{k-1} d_j & \text{otherwise} \end{cases} \tag{2.20}$$

Now, the dynamic programming algorithm (DPM1 algorithm) is defined as

$$F(i, j) = \begin{cases} 0 & \text{if } i = j \\ \varpi(j, i) & \text{if } j = 1 \\ \min_{k=j-1}^{i-1} \{F(k, j-1) + \varpi(k+1, i) & \text{if } j < i, j \neq 1 \end{cases} \tag{2.21}$$

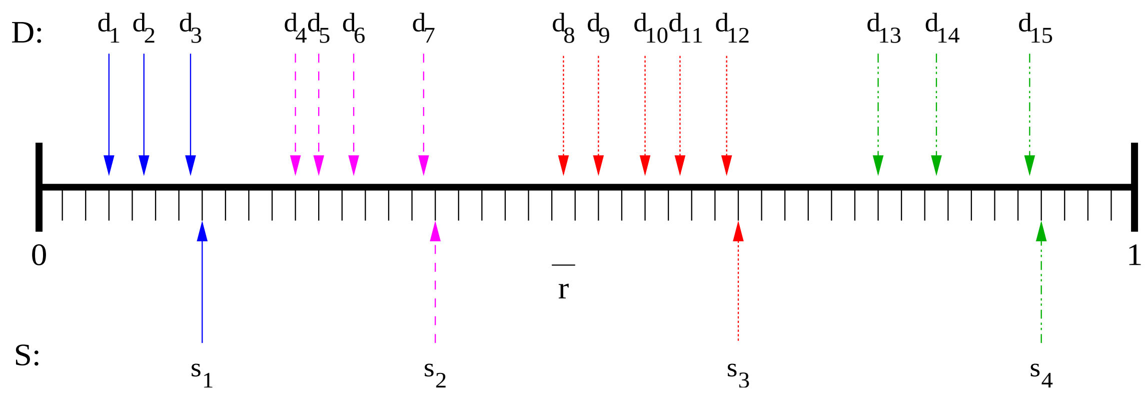

Figure 2.3: The directional p-median problem with supply points being a multiple of a basic unit r

Using the DPM1 algorithm to determine the smallest element in each coloumn in totally monotone matrices, we can solve the dynamic programming algorithm in $O(np)$. This is a substantial improvement for problem instances in which the number of demand points (in our case, the number of traffic flows) n is in the order of hundreds of thousands.

2.4 The Constrained Directional p-Median Problem

In certain applications, it is necessary to have the solution as a multiple of some basic parameter. For example, data is moved between the main memory and the disk in multiples of the page size. Also in today's high speed networks, the data transfer rate is usually a multiple of a base rate r. To this end, the directional p-median problem was extended in [3] by adding an additional constraint that all of the supply points are a multiple of some basic unit r. Hence, the new problem of quantization can be defined as follows. Given a set of demand points $D = \{d_1, d_2, \ldots, d_n\}$ find a set of p supply points $S = \{s_1, s_2, \ldots, s_p\}$ such that the total distance between each demand point and its nearest supply point is minimum and in addition, each supply point should be a multiple of some basic unit r. In other words,

$$s_i = k_i \times r, \forall s_i \in S \tag{2.22}$$

Figure 2.3 shows an instance of the constrained directional p-median problem, with a demand set D and number of supply points p identical to that in Figures 2.1 and 2.2. In this problem, we have an additional constraint that each supply point should be a multiple of a basic unit r. The mapping of the demand points to the supply points is identical to that in Figures 2.1 and 2.2, but here, the supply points are multiples of the basic unit r. Considering the Equation (2.22), the objective function $Objective(S)$ to be minimized will then be

$$Objective(S) = \sum_{j=1}^{p} \sum_{d_i \in D_j} (r \times k_j - d_i) \tag{2.23}$$

where D_j is the set of demand points mapped to the supply point s_j and r is the basic unit.

2.4.1 Solutions to the Constrained Directional p-Median Problem

Since the p supply points to be selected are multiples of r, they no longer need to be one of the demand points. So, the algorithm described in the previous section for solving the directional p-median problem was modified to take into account this fact [3]. For each value of r_i that the basic unit r can take, the p supply points for the given n demand points is determined such that all the p supply points are a multiple of r_i. The objective function $Obj_i(S)$ for each value is calculated and the optimum r_i that has the minimum objective function is chosen. The problem with this algorithm is that, for each possible value of r we need to run the DPM1 algorithm which has time complexity of $O(np)$, where n is the number of multiples of r between 0 and $1/p$. The total running time of this algorithm is $O(nxp)$, where x is the number of possible values of r. Here n and x are usually very large values. Next, we describe several heuristics for this problem which were first introduced in [3]. The various heuristics trade-off the accuracy with which the minimum objective function is found, for an improvement in the running time.

In the Demand Driven Heuristic (DDH), multiples of r_i that are closest to and greater than the n given demand points are found. These new n points would then be the potential set of supply points, thereby considerably decreasing the time complexity of the algorithm. Even though the total time complexity is still $O(nxp)$, the value of n is comparitively small here.

In the Service Driven Heuristic (SDH), the DPM1 algorithm is first used to find the p supply points for the given set of n demand points without considering the basic factor. Then, for each possible value of the basic factor r, the multiples of r that are closest to the p supply points are only considered to determine the actual set of p supply points. In Unidirectional Service Driven Heuristic (USDH), only those points that are a multiple of r and greater than the p supply points are considered. In Bidirectional Service Driven Heuristic (BSDH), this restriction is not applied. Both the heuristic approaches are faster than DDH by a factor of n, which is a significant improvement for cases where the number of demand points is large.

In the Power of Two Heuristic (PTH), the supply points do not depend on the values of the demand points. The p supply points are always the negative powers of 2 from 2^0 to 2^{p-1}. Obtaining these supply points would take only $O(p)$ time and calculating the value of of the objective function would take only $O(n)$ time. Even though this a fast heuristic compared to others, the amount of resources wasted would be considerably higher when compared to other heuristics.

2.5 Findings

A simulation study conducted by Jackson and Rouskas [9] showed that a small set of service levels is sufficient to approximate the resource usage of a continuous-rate network. In this study, six different distributions were used for generating the demand sets: uniform, triangle, increasing, decreasing, unimodal and bimodal. The results showed that, regardless of the input distribution, the amount of excess resources consumed by quantization decreases sharply as the number of service levels p increases until $p \approx 15$, beyond which the resource consumption decreases slowly. At this point, the resource consumption is no more than 5-8% resources beyond the amount requested. When the number of demands n increases, the excess resource usage also increases, but the rate of increase is slow. Overall, the results showed that demand sets of 10000 requests can be serviced by 20 or fewer service levels with no more than 5-8% excess resources consumed. Alternatively, we can say that requests up to approximately 92-95% capacity of the link bandwidth can be accepted in a quantized network.

Baradwaj [3] conducted experiments to compare the performance of the DPM1 algorithm and the heuristics described in Section 2.4.1. The results showed that DDH performed better than the other heuristics, but at the expense of higher running time. BSDH performed almost as good as DDH and PTH performed the worst. But PTH has lower running time than the other heuristics. Clearly, there is a trade-off between the performance and running time complexities of the heuristics. Also, the comparison of

DPM1 to that of DDH is interesting as the difference in the normalized costs of these two algorithms is negligible. Hence, we can say that traffic quantization incurs only a small amount of bandwidth loss.

Chapter 3

Generalized Processor Sharing and its Emulations

In this chapter, we look into the workings of Generalized Processor Sharing (GPS) and the practical scheduling techniques that emulate GPS. There are many scheduling techniques that attempt to emulate GPS: Weighted Round Robin (WRR) [5], Deficit Round Robin (DRR) [12], Generalized Virtual Clock (VC) scheduling [13], Self Clocked Fair Queueing (SCFQ) [6], Weighted Fair Queueing (WFQ) [11], Worst-case Fair Weighted Fair Queueing (WF^{2}Q) [4], among others. Out of these emulations, in this chapter we look into the two most popular ones namely, the Weighted Fair Queueing and Weighted Round Robin packet scheduling techniques in detail.

3.1 Generalized Processor Sharing

An appropriate packet scheduling discipline is important in providing Quality of Service (QoS) to the users of a network. The scheduling discipline should be flexible enough to allow different service to different users. At the same time, it should not compromise fairness. That is, one user should not receive more than the allocated service at the expense of another user. Generalized Processor Sharing (GPS) is an ideal service dicipline that provides absolute fairness among the users.

GPS is a natural generalization of uniform processor sharing. In uniform processor sharing, all the users are treated equally and they receive the same service. GPS is a work conserving service discipline, since the server is always busy, if there are packets waiting to be served. Let r denote the rate of the server and the positive real numbers $\phi_1, \phi_2, \ldots, \phi_n$ denote the weights of the N different packet flows. Let $S_i(t, t+\tau)$ be the service received by flow i at the GPS server in the interval $(t, t+\tau)$. If a flow i is continously backlogged in the interval $(t, t+\tau)$, then the GPS server satisfies the following condition

$$\frac{S_i(t,t+\tau)}{S_j(t,t+\tau)} \geq \frac{\phi_i}{\phi_j}, j = 1, 2, \ldots, N \tag{3.1}$$

In other words, the service received by the flow i relative to other flows is always greater than or equal to its relative weight. This implies that a flow which is continuously backlogged for some time τ, will receive service at least as good as it requested, for that time τ. Also, the flow i is guaranteed a rate of

$$r_i = \frac{\phi_i}{\sum_j \phi_j} r \tag{3.2}$$

The actual rate of service of the flow i (r'_i) is always greater than or equal to r_i. To show this, let $B(t)$ denote the set of backlogged flows at time t. Since the server is work conserving, the rate of service of backlogged flows r'_i could be higher than the guaranteed rate r_i in an interval (t_1, t_2) in which a set of flows are not backlogged.

$$r'_i = \frac{\phi_i}{\sum_{j \in B(t)} \phi_j} r \tag{3.3}$$

Since the number of backlogged flows is less than or equal to the total number of flows, i.e., $B(t) \subseteq \{1, 2, \ldots, N\}$, we have that

$$\sum_{j \in B(t)} \phi_j \leq \sum_j \phi_j \tag{3.4}$$

Thus, from equations (2.2), (2.3) and (2.4), we have that

$$r'_i \geq r_i \tag{3.5}$$

Once any of the non-backlogged flows becomes active at a later time, it would receive service at least as good as their guaranteed service. But none of these flows would receive preferential treatment because of the fact that they were idle during the interval (t_1, t_2). GPS scheduler maintains a seperate FIFO queue for each flow sharing the link. Unlike other queueing disciplines like the First Come First Served (FCFS) or the strict priority schemes, GPS does not bound the queueing delay of a packet based on the total queue length on its arrival. Rather it bounds the delay of the packet based on the queue length of the packet's flow.

GPS provides worst case network queueing delay guarantees and ensures absolute fairness among the flows in the network. Hence, GPS service discipline is the best suited scheme for providing QoS in the networks that serve real time traffic like voice and video. But GPS is an idealized scheme that cannot transmit packets. A GPS server serves all the backlogged flows simultaneously. It also assumes that the traffic is infinitely divisible. In realistic networks, the traffic is made of packets that cannot be broken or divided. Also, a server can provide service to only one flow at any time. An entire packet must be served before another packet can be served. Hence GPS is unimplementable in any realistic network. There are different ways of emulating GPS in networks. Weighted Round Robin (WRR) and Weighted Fair Queueing (WFQ) are two of the most popular ones.

3.2 Weighted Round Robin

Weighted Round Robin (WRR) scheduling discipline is one of the simplest emulation of the GPS discipline. WRR is a best-effort scheduling discipline that serves flows based on the weights associated with them. Round Robin scheduling discipline is a special case of WRR where all the flows have equal weights. In Round Robin discipline, the server serves one packet of each backlogged flow in every round. There are two styles of scheduling in WRR - bursty and smooth.

In the bursty service style of WRR scheduling, the server serves a specific number of packets or bytes proportional to the weight of the flow, in every round. (Since it serves a burst of packets from one flow after another, it is called bursty service.) Consider an example of 4 backlogged flows A, B, C and D with weights of 0.5, 0.2, 0.2, 0.1 respectively. Then the WRR bursty service order, for every round, would be as shown in Figure 3.1.

The advantage of this WRR scheduling style is that it is easy to implement. But this style does not provide strong fairness among the flows. The server visits a particular

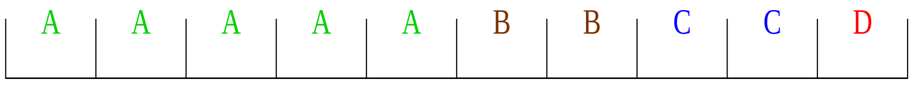

Figure 3.1: WRR bursty service for flows A, B, C and D with weights of 0.5, 0.2, 0.2 and 0.1

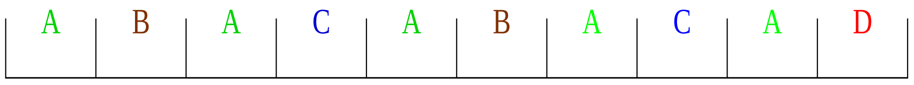

Figure 3.2: WRR smooth service for flows A, B, C and D with weights of 0.5, 0.2, 0.2 and 0.1

flow only once per round. If the flow does not have enough packets to be served in the queue, the server serves fewer number of packets than the allowed number of packets for that flow in that round, even if packets arrive at the queue after the visit by the server. For example, if there are only 2 packets of flow A in the queue, the server serves these 2 packets and moves to the next flow. If a packet of flow A arrives later in that round, the packet has to remain in the queue until the server completes the round and will be transmitted when the server visits the flow in the next round. Hence it also does not provide proper delay guarantees to the packets in the network.

In the smooth service style of WRR scheduling, packets are serviced based on service intervals. The service interval of a flow is inversely proportional to its weight. A flow is serviced only once in a service interval. After the flow is serviced in the interval, it becomes ineligible for service during the remaining time in the interval. Consider the same example of 4 backlogged flows A, B, C and D with weights of 0.5, 0.2, 0.2, 0.1 respectively. The service order, for every round, would be as shown in Figure 3.2.

In this style, the jitter is low compared to the bursty service style. But this style has high running time complexity because each time a flow becomes active or inactive, the schedule needs to be recomputed. Even though this style provides better fairness than bursty service style, it still has short-term fairness issues.

WRR does not work well with traffic that involves variable-sized packets. For variable-sized packet traffic, WRR requires the estimation of mean packet size of every flow in order to calculate the number of the packets that needs to be served in a round. In IP networks, estimating the mean packet size of the flows is difficult and hence WRR may not be able to correctly emulate GPS.

3.3 Weighted Fair Queueing

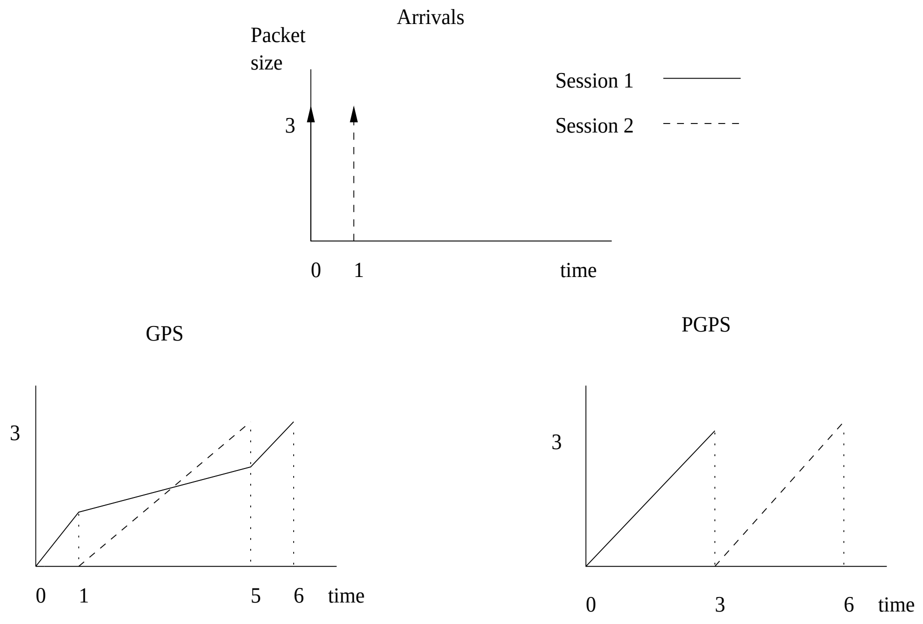

Figure 3.3: An example of GPS and PGPS service order of packets from two flows

Weighted Fair Queueing (WFQ) is the most popular emulation of the GPS. WFQ, also referred to as PGPS (Packet Generalized Processor Sharing), is a packet-by-packet transmission scheme that is a very close approximation of the GPS. Unlike WRR, WFQ works well even when the packets are of variable size.

The best emulation of GPS would be a scheduling discipline which serves packets in the same order as they would be serviced by the GPS. The packets in this scheme should be serviced in the increasing order of their finish times in the corresponding GPS system. But the problem is that the packet that needs to be serviced next may not have arrived at the system. Since the server needs to be work-conserving, it cannot remain idle if there are packets waiting to be serviced in the system. Consider an example with two flows, as shown in Figure 3.3, where $\phi_2 = 3 \times \phi_1$, and it takes three time units to service each packet. Let us assume that a session 1 packet P_1 arrives at time 0 and a session 2 packet P_2 arrives at time 1. GPS will serve session 1 packet until time 1. When the session 2 packet arrives, GPS will serve both packets simultaneously. Since session 2 has greater weight, P_2 will be served by time unit 5 and P_1 will be completely served by time unit 6. The scheme which emulates

GPS cannot serve P_2 before P_1, because P_2 does not arrive until time 1. Therefore, the best possible way to emulate GPS is to serve the first packet that would complete service in the GPS, if no additional packets were to arrive after time t. As a result, PGPS serves P_1 from time 0 to 3 and P_2 from time 3 to 6. P_2 departs the system at time 6, one time unit later than it would depart under GPS. The only packets that are delayed under PGPS are those that arrive late to be transmitted in their GPS order.

WFQ is implemented using virtual time. Virtual time $V(t)$ is used to keep track of the progress of the simulated GPS system. Consider a system with n flows which are associated with n positive real numbers $\phi_1, \phi_2, \ldots, \phi_n$, that denote their weights. Let r denote the rate of the server. Then, the following expression captures the evolution of virtual time:

$$V(t_{j-1} + \tau) = V(t_{j-1}) + \frac{\tau}{\sum_{i \in B_j} \phi_i} \tag{3.6}$$

The rate of change of V is

$$\frac{\partial V(t+\tau)}{\partial \tau} = \frac{1}{\sum_{i \in B(t)} \phi_i} \tag{3.7}$$

where $B(t)$ denotes the set of backlogged flows at time t.

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

$$S_i^k = max\{F_i^{k-1}, V(a_i^k)\} \tag{3.8}$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i} \tag{3.9}$$

The WFQ scheduler serves packets in the increasing order of their virtual finish times F_i^k.

The main advantages of WFQ are that it is a very close approximation to GPS and it provides excellent worst case network delay guarantees. Let F_p and F_p' denote the finish times of any packet p in GPS and PGPS respectively. Then,

$$F_p' - F_p \leq \frac{L_{max}}{r} \tag{3.10}$$

where L_{max} is the maximum packet length and r is the rate of the server. WFQ has excellent fairness and delay bound properties. But, the problem with the implementation of WFQ is that it has a running time complexity of $O(logN)$, where N is the total number of flows in the system. The head-of-line packets of all active flows must be in sorted order at any given time t and this operation has a time complexity of $O(logN)$. Worst-case Fair Weighted Fair Queueing (WF^{2}Q), a variant of WFQ that provides a better congestion control scheme, also has a running time of $O(logN)$.

Chapter 4

Tiered Service Fair Queueing (TSFQ)

In this chapter, we present how quantization may be applied to packet scheduling in order to overcome the time complexity problem faced by the Weighted Fair Queueing. First, we describe a basic scheduler which is suited only for traffic with fixed-size packets. We also discuss the challenges associated with this new scheduler and the solutions to overcome these challenges. Finally, we present an extension to the design of this basic scheduler that would enable it to work well for traffic with variable size packets.

4.1 TSFQ for Fixed-Size Packet Traffic

For the sake of simplicity, let us assume for the moment that all packets in the network have a fixed size L (i.e., $L_i^k = L \;\; \forall i, k$); we will remove this assumption in Section 4.2. Two components are involved in Tiered Service Fair Queueing (TSFQ): Quantization and Scheduling. The quantization part takes place during a setup or offline phase in which the optimal set of service levels are defined and the flows are mapped to appropriate service levels. The scheduling part is an online phase in which the packets are queued and serviced by the scheduler, based on the service levels set during quantization.

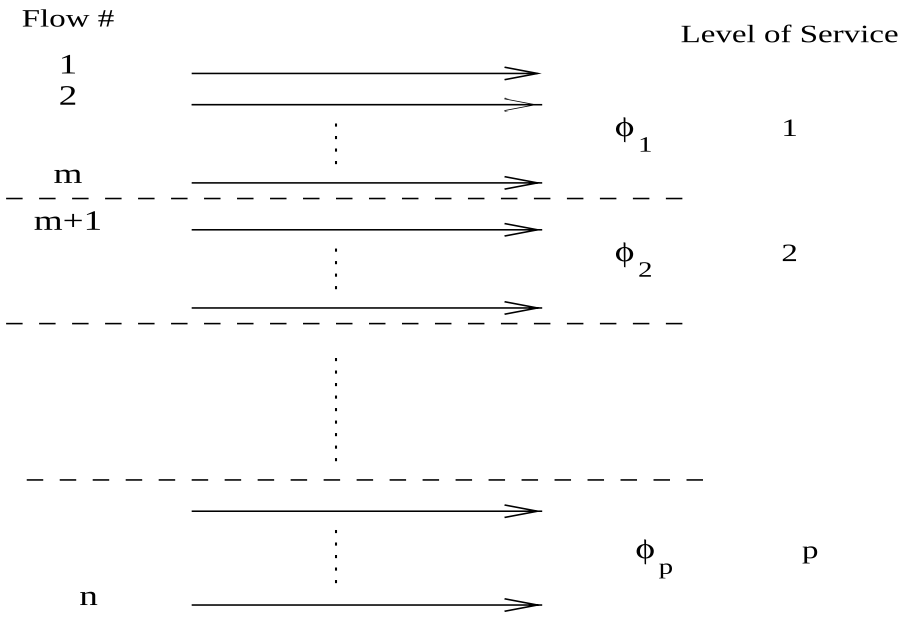

Figure 4.1: Quantization of flows: n flows mapped to p service levels

4.1.1 Quantization

A Tiered Service Fair Queueing (TSFQ) scheduler maps all the flows to service levels (tiers) based on their weights. Consider a quantized network with n flows that are mapped to a small set of p different service levels, as shown in Figure 4.1. Here, p is a constant and $p << n$. Let $\psi_1, \psi_2, \ldots, \psi_n$, denote the weights associated with the flows. Each service level l has its own weight $\phi_l,\ l = 1, 2, \ldots, p$, and all the flows that are mapped to this service level have the same weight. Each flow is mapped to its closest tier with greater weight. For any flow i that is mapped to a service level l, we have that

$$\phi_{l-1} < \psi_i \leq \phi_l,\ i = 1, 2, \ldots, n, \quad l = 1, 2, \ldots, p$$

4.1.2 TSFQ Scheduler V.1: One FIFO per Service Level

The Tiered Service Fair Queueing Scheduler maintains a number of First-In-First-Out (FIFO) queues for serving packets in an efficient manner, as shown in Figure 4.2. Similar to GPS and PGPS, the scheduler maintains a FIFO queue for each flow defined in the packet traffic. We will refer to these FIFOs as flow queues. At any time t, these flow

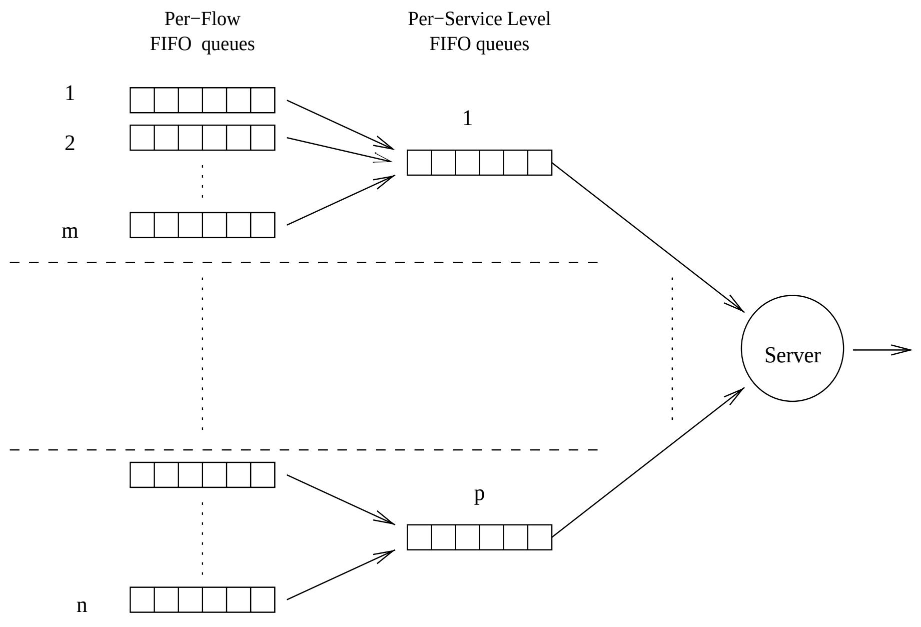

Figure 4.2: Structure of the TSFQ Scheduler V.1

queues contain the packets, that belong to their respective flows, waiting to be serviced by the scheduler. A packet arriving at a flow queue is inserted at the tail of the queue and therefore the packets within a queue are sorted in the order of their arrival times. The scheduler also maintains one FIFO queue for each service level l, $l = 1, 2, \ldots, p$. The FIFO queue at service level l contains the head-of-line packets of all the backlogged flows that are mapped to this service level.

When a new packet arrives at the head-of-line of the flow queue, it is just added to the tail of the FIFO queue. Consider a packet P arriving at time t that belongs to flow i at service level l. The packet could arrive at the head-of-line of flow i's queue in two situations. The first one is when the packet arrives at an empty flow queue and the second one is when the head-of-line packet of flow i's queue departs from the system and packet P is the next packet in the flow queue. In the first situation, the packet should be served after the head-of-line packet of the other flows at service level l that were backlogged at time t, but before all other packets of the same flows. This is because the head-of-line packets of other flow queues have arrived before the packet P and all the flows, mapped to the same service level l, have same weights. Therefore, this packet is inserted at the tail of the FIFO queue for service level l so that if the FIFO queue were in sorted order of the virtual finish

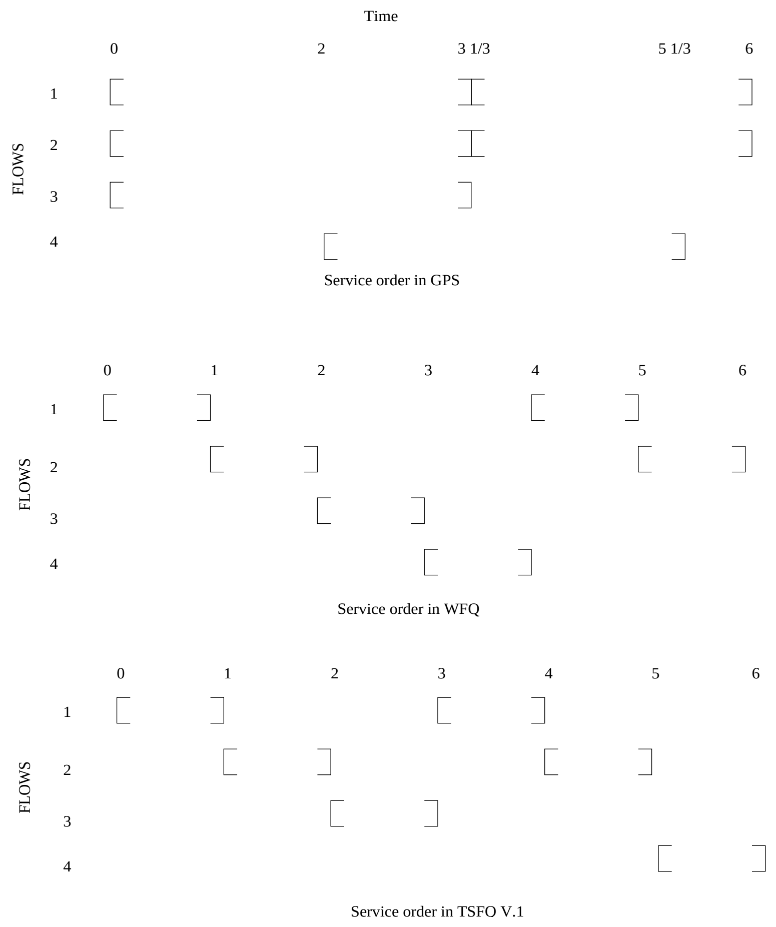

Figure 4.3: Fairness Issue Example: Comparison of Servicing Order of GPS, PGPS and TSFQ V.1

time, it will remain in sorted order after the packet insertion in most cases. We will state an exceptional case later.

In the second situation, the packet P moves to the head-of-line of the flow queue because the packet in front of it has departed the system. Again, the packet should be served after the head-of-line packets of all backlogged tier-l flows, but before any other packets of these flows. So, the packet is inserted at the tail of the FIFO queue to ensure that the FIFO queue remains sorted in increasing of the virtual finish time even after the packet insertion. Since the packets within a FIFO queue are implicitly sorted in the increasing order of the virtual finish time, the packets at each of the FIFO queues are served in FIFO order and there is no packet sorting involved. When a packet completes service, the scheduler serves the packet with the lowest virtual finish time among the head-of-line packets of the p FIFO queues. This operation involves a comparison of the virtual finish times of p packets, and, given that for a particular system p is a small constant, it takes time $O(1)$.

There is a fairness issue associated with the one-FIFO per tier scheduler. Figure 4.3 depicts this fairness issue by showing an example of servicing order of packets in GPS, PGPS and TSFQ. In this example, the quantized network has three active flows f_1, f_2 and f_3 and one inactive flow f_4 mapped to the same service level l. Also, there are more than one packet in the flow queues of f_1, f_2 and f_3 and the time required to service a packet is one time unit. After two packets, one each from flows f_1 and f_2 are served, the FIFO queue would have the new head-of-line packets of these flows at its tail. If the flow f_4, mapped to the same service level l, becomes activated when the scheduler is serving a packet from flow f_3, this new packet from f_4 would be inserted at the tail of the FIFO queue. This packet from flow f_4 would be served after the packets from flows f_1 and f_2 are served. This is unfair because the virtual finish times of the packets from flows f_1 and f_2 are greater than the virtual finish time of the packet from f_4 in the emulated GPS system. The packets from flows f_1 and f_2 start receiving service before its virtual start time. In this example, the service time of the packet from f_4 is delayed by the sum of service times of the two packets from f_1 and f_2. In the worst case, the service time of a packet in a TSFQ scheduler can be delayed from its virtual finish time by as much as the sum of $n-2$ packets' service times, where n is the total number of flows associated with the scheduler.

Let us analyze the worst-case delay performance of this first version of TSFQ. A class of generalized Guaranteed Rate (GR) scheduling algorithms that includes work conserving algorithms like generalized Virtual Clock, Packet-by-Packet Generalized Processor Sharing (PGPS) and Self Clocked Fair Queueing (SCFQ) algorithms, was defined in [7]. The class of guaranteed rate scheduling algorithms was defined to include algorithms that guarantee that a packet would be transmitted by its guaranteed rate clock (or deadline) value plus some constant. To define the guaranteed rate clock (GRC) value, let p_f^j and l_f^j denote the j^{th} packet of flow f and its length, respectively, and let $r_f^{j,i}$ be the rate allocated to packet p_f^j at server i. Also, let $A^i(p_f^j)$ denote the arrival time of packet p_f^j at server i. Then, the guaranteed rate clock value of p_f^j at server i, denoted by $GRC^i(p_f^j, r_f^{j,i})$, is given as:

$$GRC^i(p_f^j, r_f^{j,i}) = max(A^i(p_f^j), GRC^i(p_f^{j-1}, r_f^{j-1,i})) + \frac{l_f^j}{r_f^{j,i}} \tag{4.1}$$

where $GRC^i(p_f^0, r_f^{0,i}) = 0$. Let $L_{SA}^i(p_f^j)$ denote the departure time of the packet p_f^j at server i using the scheduling algorithm SA. Then, we can formally state that if

$$L_{SA}^i(p_f^j) \leq GRC^i(p_f^j, r_f^{j,i}) + \beta^i, \tag{4.2}$$

then the scheduling algorithm $SA \in GR$ class. Here, β^i is a constant denoting the worst case delay. GPS is an ideal and unimplementable system with constant factor $\beta^i = 0$. As we have shown in Section 3.3, WFQ, in the worst case, will delay a packet's departure from its GRC value by the service time of the maximum sized packet at server i. WFQ also belongs to GR class, with $\beta^i = \frac{l_{max}^i}{C^i}$, where C^i is the capacity of the server. Previously in this section, we showed that this version of TSFQ, in the worst case, will delay the departure of a packet from its GRC value by as much as the sum of $n-2$ packets' service time, where n is the total number of flows associated with the server. Regardless of the fact that the constant β^i value of TSFQ is higher compared to WFQ, this first version of TSFQ also provides delay guarantee for all the packets flowing through the scheduler and hence it belongs to the Guaranteed Rate (GR) class. Formally,

$$L_{TSFQ}^i(p_f^j) \leq GRC^i(p_f^j, r_f^{j,i}) + \sum_{k=1}^{n-2} \frac{l_k^{max}}{C^i} \tag{4.3}$$

where l_k^{max} is the maximum length for packets in flow k.

4.1.3 TSFQ Scheduler V.2: TWO FIFOs per Service Level

In order to overcome the fairness problem of the TSFQ scheduler V.1, we define a two-FIFOs per service level TSFQ scheduler. Figure 4.4 shows the structure of this new scheduler. The scheduler maintains two FIFOs for each service level and only one FIFO remains active at any given time. In this two-FIFO scheduler, the head-of-line packets of a newly activated flow are always sent to the active FIFO and the head-of-line packets of previously backlogged flows are always sent to the inactive queue. The scheduler always serves packets from the active FIFO queue. When there are no more packets to be served in the active FIFO queue, the scheduler switches to the inactive queue and serves packet from it, making it active. The other FIFO becomes inactive at this moment. By sending the head-of-line packets of previously backlogged flows to the inactive queue, we ensure that

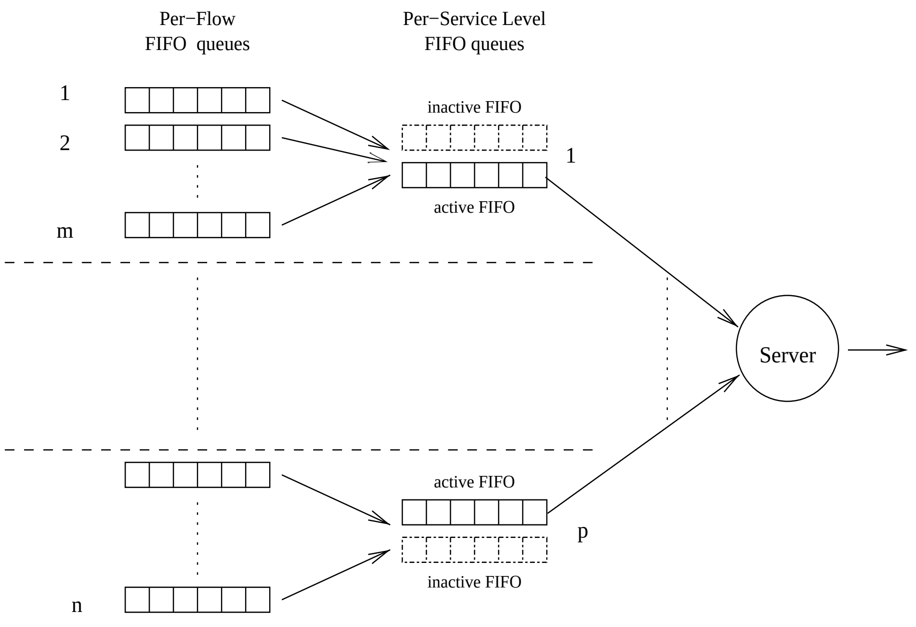

Figure 4.4: Structure of the TSFQ Scheduler V.2

the packets do not start receiving service before their virtual start time, thereby overcoming the fairness problem of the TSFQ V.1 scheduler.

Even though the two-FIFO scheduler solves the fairness problem faced by the one-FIFO scheduler, the addition of another FIFO introduces a new fairness problem. To understand this, consider a system of 3 flows, f_1, f_2 and f_3, assigned to a tier l, as shown in the Figure 4.5. Let us assume that it takes one unit time to service a packet. At time 0, let us assume that the active FIFO queue has one packet from flow f_1 and this packet is being serviced by the scheduler. Also let us assume that the inactive FIFO queue is empty. If at time 1/2 a packet from flow f_2 arrives at the scheduler, it would be added to the tail of the active FIFO. When the packet from flow f_1 finishes its service at time 1, the next packet in its flow queue will be added to the inactive FIFO. If a new packet from flow f_3 arrives just before the departure of the packet from f_2, it will be added to the tail of the active FIFO. Then, this new packet will be served immediately after the packet from f_2 and before the packet from f_1, which is in the inactive FIFO queue. But, we can see from the figure that GPS services the packet from f_1 before the packet from f_3.

The algorithm for the operation of two-FIFO scheduler includes two procedures. The **Arrival** procedure defines the action that needs to be taken on arrival of a packet.

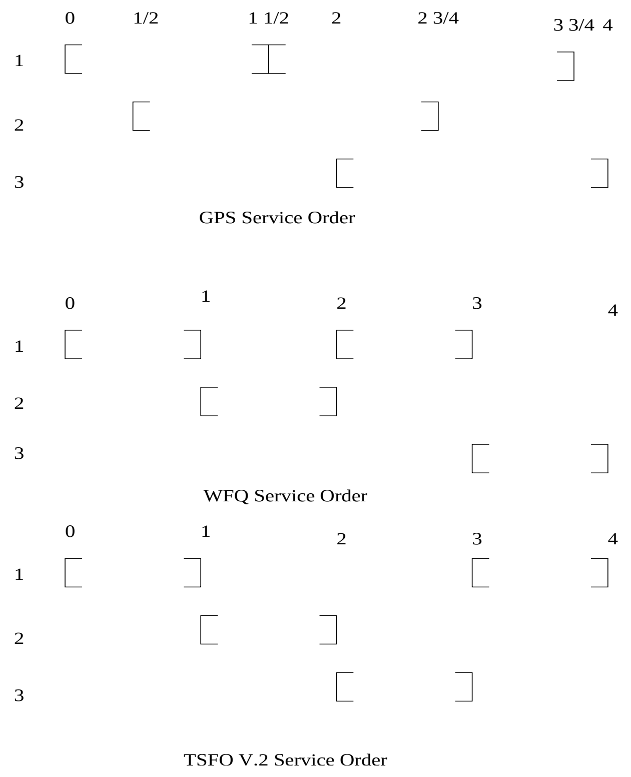

Figure 4.5: Fairness Issue Example: Comparison of Servicing Order of GPS, PGPS and TSFQ V.2

The **Departure** procedure defines the steps involved in the departure of a packet.

4.1.4 TSFQ Scheduler V.3

In this section, we define a new version of TSFQ scheduler which overcomes the problems of other TSFQ versions. TSFQ V.3 solves the fairness problems by ensuring that insertion of packets into the FIFO queue only occurs at the virtual start time of the packet's service in the emulated GPS system. This version of TSFQ scheduler operates by maintaining a GPS queue. For each active flow in the packet scheduler, the GPS queue maintains its flowid and the finish time of its head-of-line packet in the emulated GPS system. This queue contains only the information about the packet and not the actual packets. A seperate GPS queue is maintained for each service level. Like TSFQ V.1, this version of TSFQ scheduler also maintains only one FIFO and any newly actived flow's packet is directly inserted into the FIFO queue, but unlike TSFQ V.1 the next packet in

Algorithm 1 Arrival(pkt)

```
fid ⇐ flow id of arriving pkt
tierid ⇐ tier id of arriving pkt
calculate virtual finish time of pkt
if flow_queue[fid] is empty then
  if fid is marked inactive then
    insert pkt into the tail of active tier_queue[tierid]
  else
    insert pkt into the tail of inactive tier_queue[tierid]
  end if
else
  insert pkt into the tail of flow_queue[fid]
end if
```

Algorithm 2 Departure

```
pkt ⇐ packet with minimum virtual finish time in the k active tier queues
fid ⇐ flowid of pkt
tierid ⇐ tierid of pkt
if flow_queue[fid] is not empty then
    new_pkt ⇐ head-of-line packet of flow_queue[fid]
    insert new_pkt into the tail of inactive tier_queue[tierid]
else
    mark fid as active
end if
if tier_queue[tierid] is empty then
    mark all flows belonging to tierid as inactive
    swap active and inactive tier_queue[tierid]
end if
remove and return pkt
```

the flow queue of the departed packet is not added immediately to the tail of FIFO during the departure event. It is rather added only when the packet departs in the emulated GPS system. An event is scheduled at the virtual finish time in the head-of-line element of the GPS queue. When this event is dispatched, the head-of-line value of the GPS queue is removed and the virtual finish time of next packet in the flow queue, if any, is added to the tail of GPS queue and the next event scheduled. Also at this point, we add the next packet in the flow queue, if any, to the tail of the FIFO queue. From equation 3.8, we can say that the packets are inserted into the FIFO queue only at their virtual start time.

Lemma 1 *Out of all currently available packets, TSFQ V.3 scheduler always serves the one with the least virtual finish time.*

Proof: In TSFQ V.3 scheduler, packets are inserted at the tail of the FIFO queue at their virtual start time. So, the packets are arranged with in the FIFO queue in the order of their virtual start time and are also served in the order of virtual start time with in a single service level. Since the packets within a single service level FIFO have same weights and same packet size, we can conclude from equation 3.9 that the packets are also served in the order of their virtual finish time within the service level. Since the scheduler picks the packet with the least virtual finish time among the head-of-line packets of all the service level FIFOs, for service, we can conclude that TSFQ V.3 scheduler serves the packet with the least virtual finish time among all the currently available packets. □

4.2 TSFQ for Variable-Size Packet Traffic

In a network with variable-sized packets, p FIFO queues, one for each service level, are not sufficient enough for constant time operation. However, we can exploit the fact that in the Internet, certain packet sizes dominate [10], so as to implement TSFQ efficiently. As shown in Figure 4.6, the scheduler may maintain k seperate FIFOs for each service level,

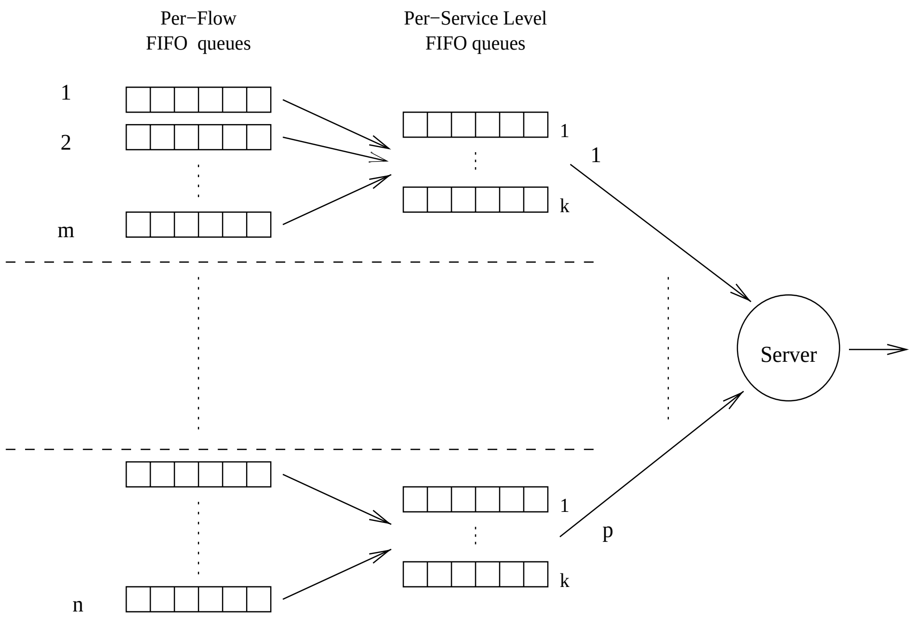

Figure 4.6: Structure of the TSFQ Scheduler for Variable-Size Packet Traffic

for a total of pk FIFO queues, where k is a small constant based on the number of common packet sizes. Some of the k FIFOs within a level are used for packets with common packet sizes like 40, 576, 1500 bytes, etc. The remaining FIFOs are used for the packets with sizes between these common values. For example, one queue for packets of size less than 40, another for size 41-575 bytes, another for size 577-1499 bytes and another for packets of size greater than 1500 bytes. In this case, $k = 7$. For a two-FIFO scheduler, we require two FIFOs for each set of packet sizes within each service level. So, there will be $2k$ FIFO queues for each service level, for a total of $2pk$ FIFO queues.

The queues dedicated to common packet sizes do not require any sorting. They operate identically to the FIFO queues in a network with fixed-size packets. However, the queues dedicated to packets with size between the common values must be sorted in the non-decreasing order of virtual finish time of the packets, at the time of packet insertion. Since more than 90% of the internet traffic consists of packets with common sizes [10], no sorting operations are necessary for the large majority of packets. The sorting operations take place infrequently (less than 10% of the time) and involves only relatively short queues, since less than 10% of the packets are spread over several queues at l different service levels. The time complexity of the sorting operations depend only on the network load and the ratio of packets with a non-common size, and is independent of the number n of flows. Finally, the scheduler serves the packet with least virtual finish time among the head-of-line packets of pk queues, where pk is constant for a given system.

Chapter 5

ns Simulations

In this chapter, we turn our attention to the implementation details of the different scheduler versions proposed in the previous chapter. The schedulers were either implemented or modified in the Network Simulator $ns-2$. We start with a brief introduction to the simulator $ns-2$ and then we deal with the implementation of the schedulers in $ns-2$.

5.1 $ns-2$ Simulator

ns is a discrete event simulator, developed at the Lawrence Berkeley Labaratory (LBL) of the University of California, Berkeley (UCB) and extended and distributed by the VINT project (a collaboration between University of Southern California (USC)/Information Sciences Institute (ISI), LBL/UCB and Xerox PARC). *ns* is targeted mainly at networking research and education. *ns* is widely used by the networking community and is widely regarded as a critical component of research infrastructure. *ns* provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

ns is an object oriented simulator, written in C++, with an OTcl (an object oriented version of Tcl) interpreter as a front-end. *ns* uses the split programming paradigm because the simulator is designed to do two different kind of things. On the one hand,

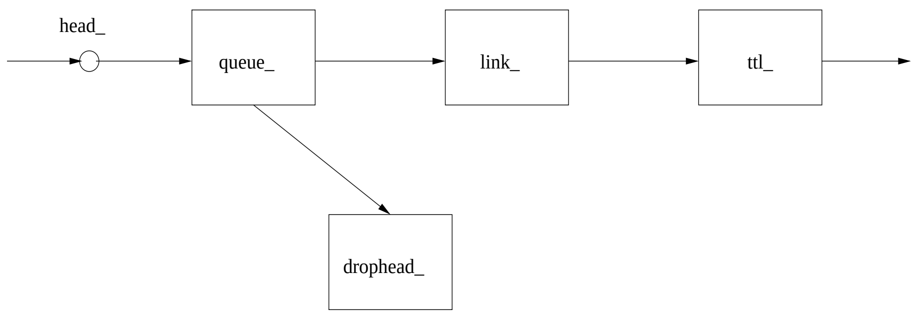

Figure 5.1: Composite Construction of a Unidirectional Link in *ns*

detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. In these cases, iteration time (changing the model and re-running the simulation) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. The combination of C++ and OTcl meets both of these needs. C++ is fast to run but slower to change, making it ideal for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. TclCL (Tcl with classes) provides a layer of C++ glue over OTcl, making objects and variables appear in both languages.

5.1.1 Packet Scheduling in $ns-2$

In ns, a simple link is built from a sequence of connectors. A connector will receive a packet, perform some function, and deliver the packet to its neighbour, or drop the packet. ns provides the instance procedure $simplex-link$ to create a unidirectional link from one node to another. The syntax for creating a simplex link is:

$$\$nssimplex-link\{from-node\}\{to-node\}\{bandwidth\}\{delay\}\{queue-type\}$$

This command creates a simplex link from node $<from-node>$ to node $<to-node>$ with specified $<bandwidth>$. $<delay>$ specifies the propagation delay of the link and a queue of type $< queue - type >$ is installed for the link. As shown in Figure 5.1, five instance-variables/connectors define a simple unidirectional link:

head_ Entry point to the link, it points to the first object in the link;

queue_ Reference to the main queue element of the link;

link_ Reference to the element that actually models the link, in terms of the delay and bandwidth characteristics of the link;

ttl_ Reference to the element that manipulates the ttl (time to live) in every packet;

drophead_ Reference to an object that is the head of a queue of elements that process link drops.

Out of these five instance variables, the queue component is the object of interest to us.

ns currently supports drop-tail (FIFO) queueing, RED buffer management, Class Based Queueing (CBQ), and variants of Fair Queueing including, Fair Queueing (FQ), Stochastic Fair Queueing (SFQ), and Deficit Round-Robin (DRR). Other Scheduling disciplines like Weighted Fair Queueing (WFQ), RED In/Out(RIO) and Core-Stateless Fair Queueing (CSFQ) are available as seperate modules in the Internet. The *Queue* class is the base class of all the queues in *ns*. This class is used by particular types of (derived) queue classes. The class includes two important member functions: *enque* and *deque*. Both these functions are pure virtual, indicating that the particular queues derived from *Queue* must implement these two functions. Other member functions are usually not overridden in the derived class.

When a *Queue* object receives a packet, it calls the subclass (queueing discipline specific) version of the *enque* function with the packet. The *enque* function determines how the packets are stored in the queue. Transmission delays are simulated by blocking the queue until it is re-enabled by its downstream neighbor. When a queue is blocked, it is able to enque packets but not send them. If the queue is not blocked, it is allowed to send packets and calls the specific *deque* function which determines which packet to send, blocks the queue (because a packet is now in transit), and sends the packet to the downstream neighbor. Any future packets received from upstream neighbors will arrive to a blocked queue. When the downstream neighbor wishes to unblock the queue after the transmission delay, it invokes the *resume* function of the queue object, which will in turn invoke *deque* to send the next scheduled packet downstream (and leave the queue blocked). If there is no packet ready to be sent in the queue, the queue is unblocked.

5.2 TSFQ Implementation in $ns - 2$

To test the performance of the TSFQ schedulers, both the versions were implemented in the network simulator $ns-2$. Since the objective of designing the TSFQ scheduler was to create a scheduler with fairness as good as that of WFQ and running time complexity as good as that of WRR, we chose WRR and WFQ for comparing the performance of TSFQ scheduler. $ns - 2$ package includes a Smooth service sytled Weighted Round Robin (SRR) implementation. WFQ implementation for $ns - 2$, contributed by Paolo Losi [1], is available as a separate patch in the *ns* website. This implementation of WFQ has an infinite queue size and the scheduler never drops a packet. To avoid this, the source code of WFQ implementation was modified to limit the size of the queue. In this modified version of WFQ, the shared buffer size of the queue can be limited in terms of either packets or bytes. One flow may overwhelm the entire queue by sending packets at a high rate. This will affect the other flows since their packets will be dropped when the shared buffer size exceeds the limit. To overcome this problem, each flow is assigned a separate limit based on their weight. When the queue limit is exceeded, only the packets belonging to the flow that has exceeded its own limit is dropped.

TSFQ and TSFQAggregClassifier are the two important classes that define the functioning of the TSFQ. TSFQ class, derived from the base class *Queue*, is the main queue class of the TSFQ implementation. A TSFQ object is instantiated when a TSFQ queue is defined on a link connecting two nodes. Every TSFQ object defined in the network is associated with a TSFQAggregClassifier object. TSFQAggregClassifier object holds the necessary details for the functioning of the queue. It holds the following details of a TSFQ object:

- The weights associated with the queues (flow-queues) defined within the TSFQ scheduler.
- The weights associated with the service levels defined in the TSFQ scheduler.
- The flow to queue mapping of all the flows passing through the TSFQ scheduler.

- The flow to service level mapping of all the flows.

The TSFQAggregClassifier class implements all the functionality required for the appropriate initialization and manipulation of the TSFQ objects. The TSFQAggregClassifier object should be created during simulation initialization, which should be followed by assigning weights to flows and service levels. The following is the syntax for assigning weights to flows and service levels:

1. $\$ < tsfqclassifier - object > setqueue < flow - id >< queue - id >$
2. $\$ < tsfqclassifier - object > setweight < queue - id >< weight >$
3. $\$ < tsfqclassifier - object > settier - weight < tier - id >< weight >$

Line 1 maps a flow to a queue, while lines 2 and 3 assigns a weight to a queue and a service level respectively.

The TSFQ object is the actual queue object placed before the link. The TSFQ object receives the packets, that needs to be transmitted on the associated channel, enqueues and serves them appropriately. The packets may be received from an upstream node or generated by the node itself. The TSFQ object holds a reference to its associated TSFQAggregClassifier object. A reference to the TSFQAggregClassifier object is passed to the TSFQ object using the *install* Tcl command. The syntax for *install* command:

$$\$ns\ tsfqclassifier - install\ \ < from - node > < to - node > < tsfqclassifier - object >$$

The pure virtual functions of the base class *Queue*, *enque* and *deque*, are implemented in the TSFQ class. The *enque* function is automatically invoked by the arrival of a packet at the scheduler. The *deque* function is also automatically invoked when the outgoing link becomes idle. The flow queues and service-level FIFOs are implemented as linked list data structures. Each element in the list structure represents a packet in the FIFO. Each element contains a reference to the packet it represents and its associated virtual finish time.

The *enque* method inserts the packet in the corresponding flow queue. If the packet is the head-of-line packet of the flow queue, then a reference of the packet is also inserted at the tail of the associated service level queue. The *enque* method also calculates the virtual finish times of all the packets that arrive at the queue. The *deque* method compares the virtual finish times of all the head-of-line packets and determines the packet with the least virtual finish time. The packet is removed from the list structure and returned. If flow queue of the packet is not empty, the head-of-line packet of the flow queue is inserted to the tail of the service level queue. The TSFQ class also includes methods to keep track of the GPS virtual time. The *schedule* method is invoked at the virtual arrival and virtual departure of a packet from the scheduler. Virtual arrival denotes the arrival of a packet at the service-level FIFO and Virtual departure denotes the departure of a packet in the emulated GPS system. The *schedule* method schedules an event at the least virtual finish time. The event handler function updates the virtual time of the scheduler by virtually departing the packet with the least virtual finish time.

Chapter 6

Numerical Results

We now present a set of numerical results comparing the performance of several packet scheduling techniques discussed in Chapters 3 and 4:

1. WFQ
2. WRR - Smooth
3. TSFQ V.1
4. TSFQ V.2
5. TSFQ V.3

We consider four different scenarios which are discussed in each of the following subsections. For all the scenarios, the topology we used is shown in Figure 6.1. Nodes 1 - 4 are the source nodes, nodes 7 - 10 are the destination nodes, and nodes 5 and 6 are the intermediate nodes. Here, the queue of interest is the queue at node 5. The outgoing link associated with this queue has a bandwidth of 2Mbps.

In our simulations, we used three arrival processes: Constant Bit Rate (CBR), Exponential and Pareto with a shape parameter of 1.5. Each flow is associated with a single arrival process, and its packet arrivals are generated according to this process:

- *CBR flows* are characterized by their average rates.

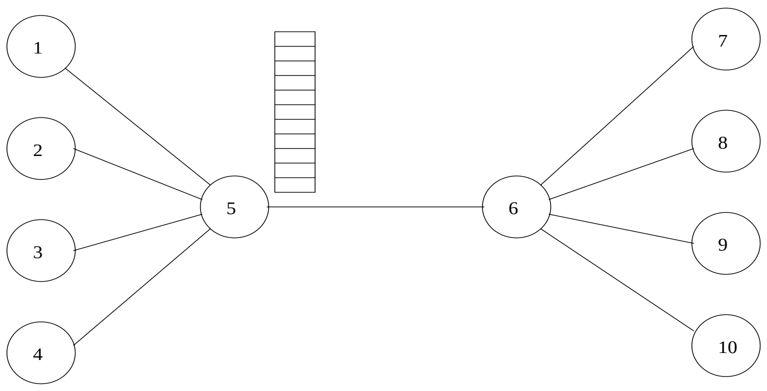

Figure 6.1: Topology used for the simulations

- *Exponential flows* are characterized by their average rates and the percentage of their on/off times.
- *Pareto flows* are also characterized by their average rates and the percentage of their on/off times.

We use the following peformance metrics to characterize and compare the performance of the various packet scheduling algorithms:

1. *Packet departure times*: We are interested in showing that TSFQ scheduler versions serve packets in an order similar to that of WFQ, provided that the quantized weights of the flows in TSFQ are the same as the weights of flows in WFQ. Hence, we measure the packet departure order of various versions of TSFQ and WRR relative to WFQ. To compute the departure order of the schedulers relative to WFQ, individual departure times of 100 packets were noted for each scheduler sch and then the relative departure of the scheduler $rdep_{sch}$ was calculated using the formula

$$rdep_{sch} = dep_{sch} - dep_{WFQ} \tag{6.1}$$

   where dep_{sch} and dep_{WFQ} denote the individual departure times of packets under the schedulers sch and WFQ respectively.

2. *Short-term throughput*: This performance metric characterizes the fairness property of the packet scheduling algorithms. Since we are interested in showing that TSFQ

provides better fairness among flows than WRR, we measured the short-term throughput of the packet scheduler. The short-term throughput of a flow was computed by measuring the amount of its bytes served by the scheduler over an interval of 50ms. The relative short-term throughput of the flow served by a scheduler sch ($RSTTP_{sch}$) is then computed using the formula

$$RSTTP_{sch} = \frac{STTP_{sch}}{STTP_{WFQ}} \tag{6.2}$$

where $STTP_{sch}$ and $STTP_{WFQ}$ denote the short-term throughputs of the flow under the schdulers sch and WFQ respectively.

3. *Average and worst-case delay*: Since delay is an important performance measurement for any QoS-aware network, we measured the average and worst-case delay of the flows served by the schedulers.

The following subsections describe each experimental scenario and discusses the results in detail.

6.1 Fixed-Size Packets with a Single Service level

In this experiment, we consider fixed-size packet traffic, and we assume that all flows are assigned to the same service level with weight ϕ. We simulated 10 flows, which were classified as follows:

- five flows are CBR flows with average rates of 220 Kbps, 350 Kbps, 240 Kbps, 280 Kbps and 160 Kbps.
- three flows are exponential flows with average rates of 190 Kbps, 200 Kbps and 130 Kbps and on/off times of 0.7/0.3, 0.25/0.75 and 0.5/0.5.
- two flows are pareto flows with average rates of 400 Kbps and 200 Kbps and on/off times of 0.5/0.5 and 0.7/0.3.

All the flows generate packets of size 210 bytes and have same weights in all the scheduling techniques.

Figure 6.2 shows the relative departure of packets from all flows and Figures 6.3 and 6.4 show the relative departure of packets from a single flow. Since the measure is

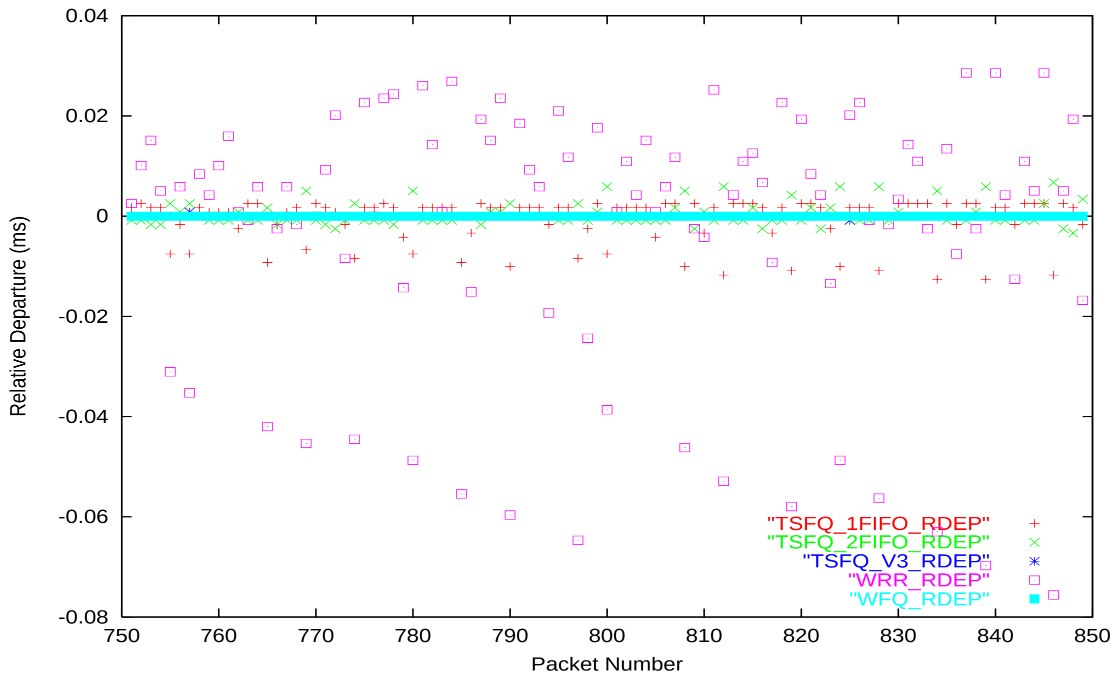

Figure 6.2: Relative Departure - Single service level, Fixed-size packets - over all flows

relative to WFQ, the relative departure value of WFQ is always zero. From these figures, we can see that TSFQ V.3 has the same departure order as that of WFQ. These figures also show that the TSFQ versions 1 and 2 differ from WFQ by only a small amount compared to WRR.

Figures 6.5 and 6.6 show the short-term throughput of two flows under different scheduling schemes, relative to WFQ. Since the measure is relative to WFQ, it is always one for WFQ. It can be seen from these figures that the relative short-term throughput of WRR exhibit heavy fluctuations, thereby indicating poor fairness among flows. In WRR, a flow may significantly under perform or over perform. Even though TSFQ versions 1 and 2 exhibit fluctations, they are smaller compared to WRR. Since TSFQ version 3 serves packets in the same order as that of WFQ, its relative throughput is also always one.

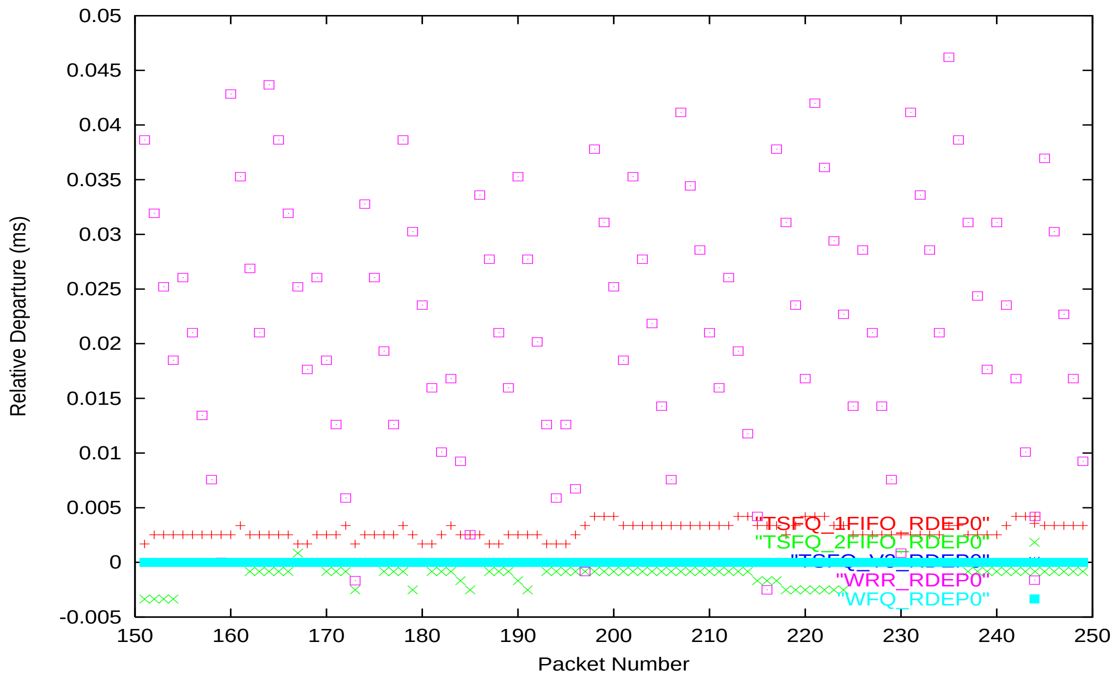

Figure 6.3: Relative Departure - Single service level, Fixed-size packets - over Flow 1

6.2 Fixed-Size Packets with Multiple Service Levels

We conducted two sets of experiments for this scenerio, one with a small set of flows and one with a larger set of flows.

6.2.1 Small Set of Flows

In this experiment, we consider fixed-size packet traffic of 20 flows mapped to 5 different service levels with weights 0.05, 0.1, 0.15, 0.25 and 0.4. Each service level has different number of flows mapped to it, with 8 flows getting mapped to service level with weight 0.05, 7 flows mapping to service level with weight 0.15, 4 flows mapping to service level with weight 0.25 and the remaining one flow getting mapped to service level with weight 0.4. The flows generate packets of size 210 bytes and they can be classified as follows:

- ten flows are CBR flows with average rates of 110 Kbps, 175 Kbps, 140 Kbps, 120 Kbps, 200 Kbps, 80 Kbps, 100 Kbps, 150 Kbps, 90 Kbps and 160 Kbps.

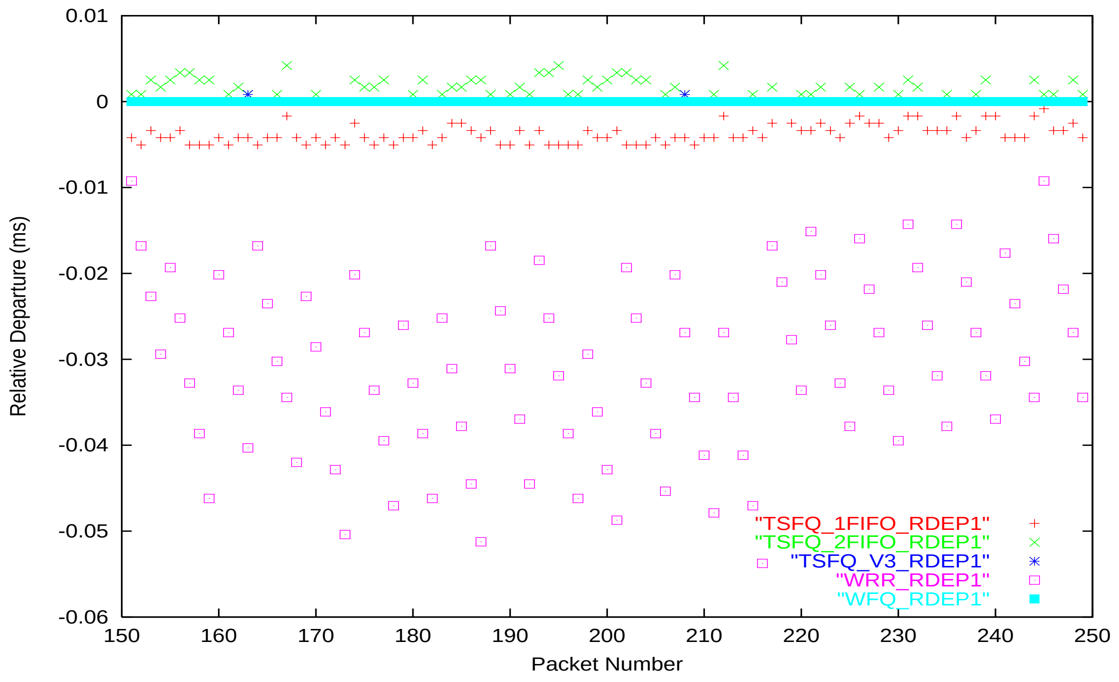

Figure 6.4: Relative Departure - Single service level, Fixed-size packets - over Flow 2

- seven flows are exponential flows with average rates of 90 Kbps, 100 Kbps, 50 Kbps, 80 Kbps, 200 Kbps, 110 Kbps and 130 Kbps and on/off times of 0.7/0.3, 0.5/0.5, 0.8/0.2, 0.7/0.3, 0.2/0.8, 0.5/0.5 and 0.25/0.75.

- three flows are pareto flows with average rate of 150 Kbps and 200 Kbps and on/off times of 0.4/0.6 and 0.5/0.5.

Figure 6.7 shows the relative departure of packets from all flows and Figures 6.8 and 6.9 show the relative departure of packets from a single flow. Figures 6.10 and 6.11 show the short-term throughput of two flows under different scheduling schemes, relative to WFQ. From the figures, we see that there is a slight change in the departure order of packets between TSFQ V.3 and WFQ. This because, WFQ serves packets in the order of their virtual finish time whereas TSFQ V.3 serves packets within a single service level in the order of their virtual start time and is equivalent to WF^2Q within a service level. Other observations from these figures are similar to those in the previous section.

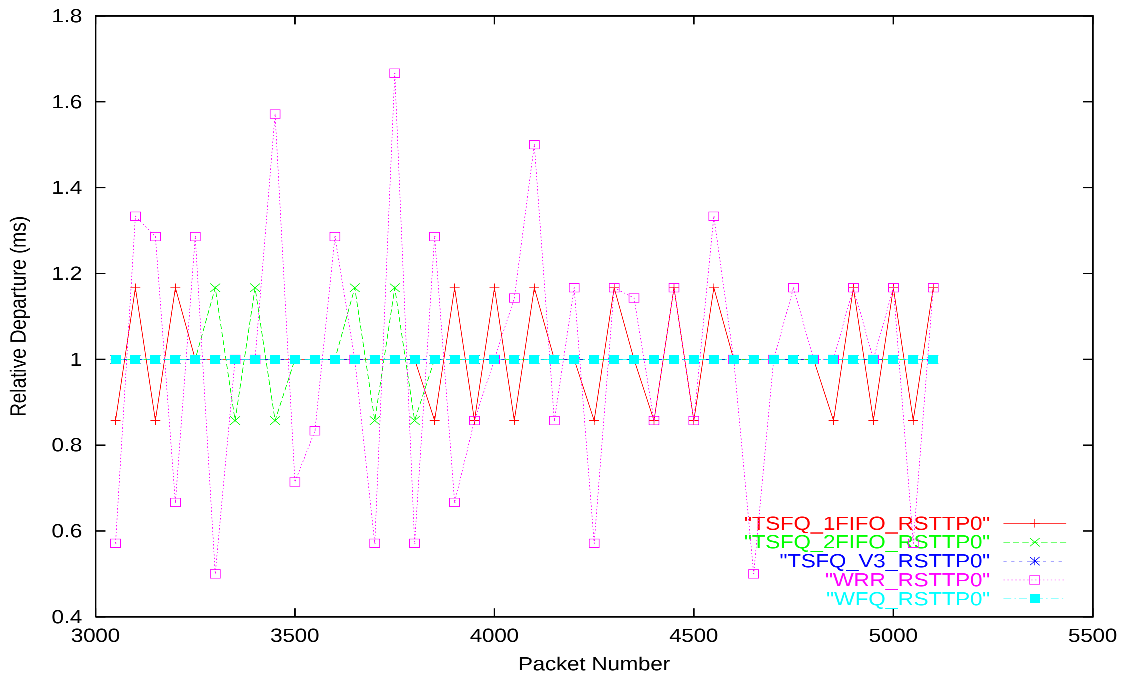

Figure 6.5: Relative Throughput - Single service level, Fixed-size packets - Flow 1

6.2.2 Large Set of Flows

In this second experiment, we simulate a fixed-size packet traffic of 1000 flows mapped to 10 different service levels. Each service level has different number of flows mapped to it and each flow has different set of parameters. The average rates of the flows vary from 1 kbps to 200 kbps. Figure 6.12 shows the relative departure of packets from all flows and Figures 6.13 and 6.14 show the relative departure of packets from a single flow. Figures 6.15 and 6.16 show the short-term throughput of two flows under different scheduling schemes, relative to WFQ. The observations from these figures are similar to those in the previous section.

6.3 Variable-Size Packets with a Single Service Level

In this experiment, we consider variable-size packet traffic, and we assume that all flows are assigned to the same service level with weight ϕ. We simulated 10 flows, similar to those described in the Section 6.1 in terms of arrival processes and average rates. But

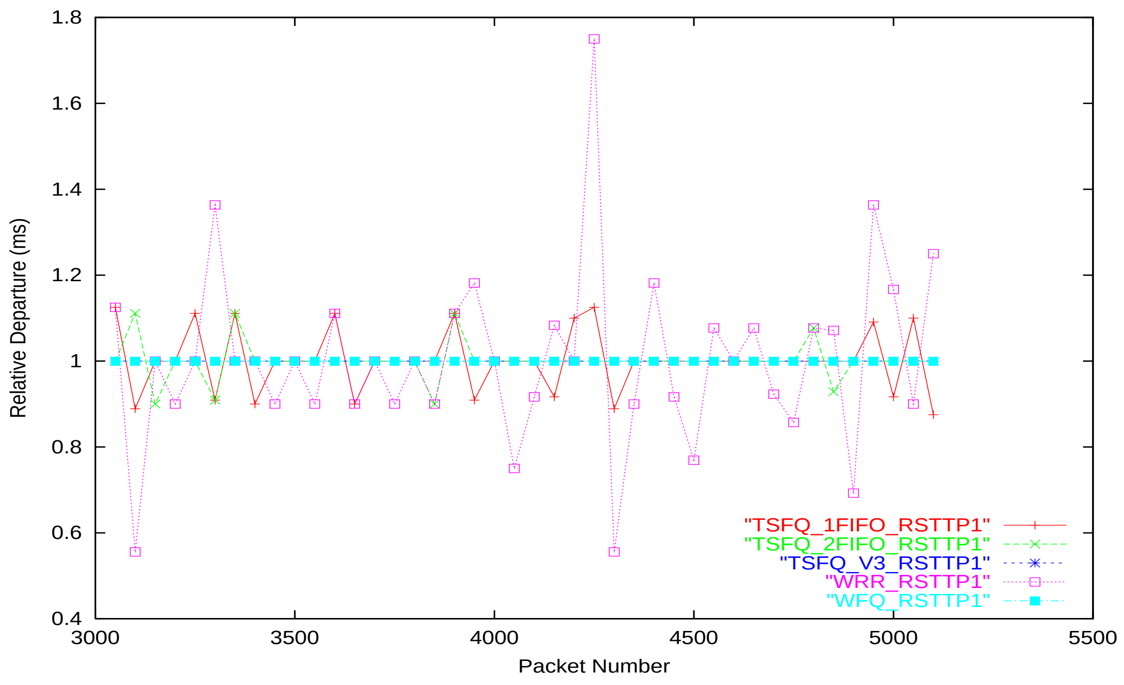

Figure 6.6: Relative Throughput - Single service level, Fixed-size packets - Flow 2

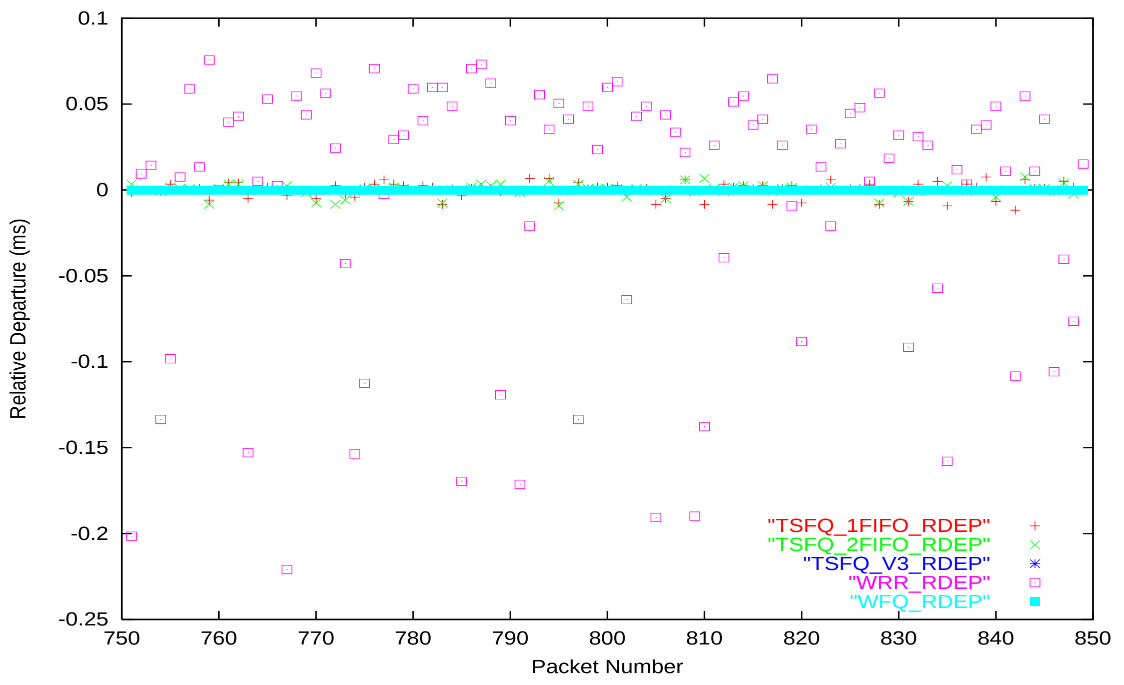

Figure 6.7: Relative Departure - Multiple service levels, Fixed-size packets - 20 flows - over all flows

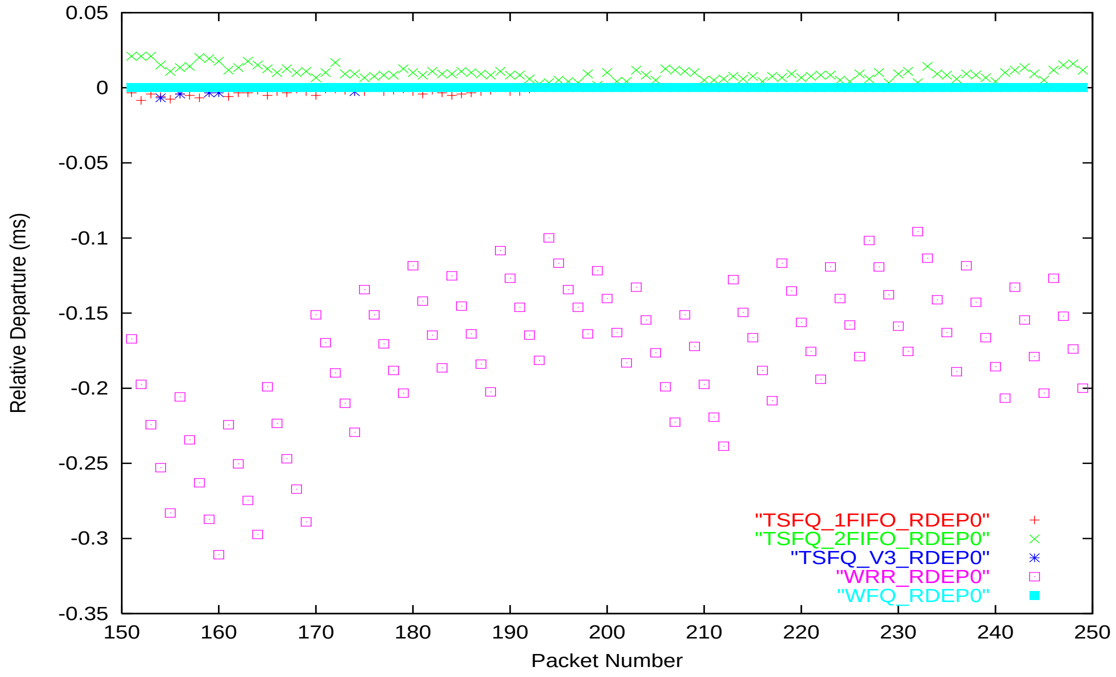

Figure 6.8: Relative Departure - Multiple service levels, Fixed-size packets - 20 flows - over Flow 1

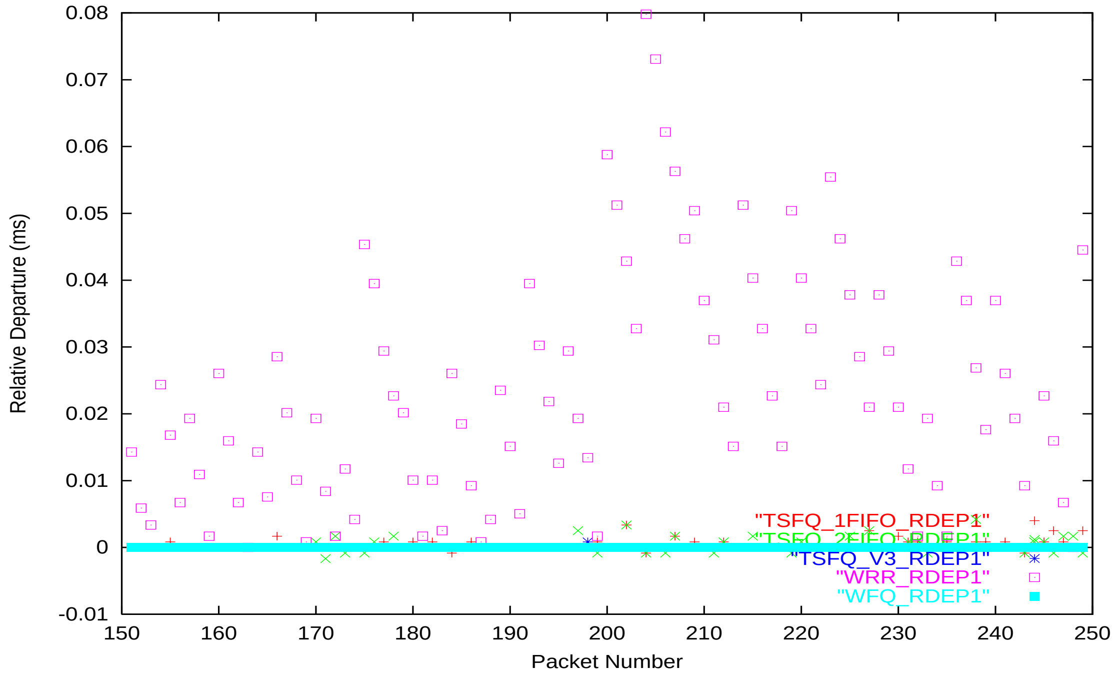

Figure 6.9: Relative Departure - Multiple service levels, Fixed-size packets - 20 flows - over Flow 2

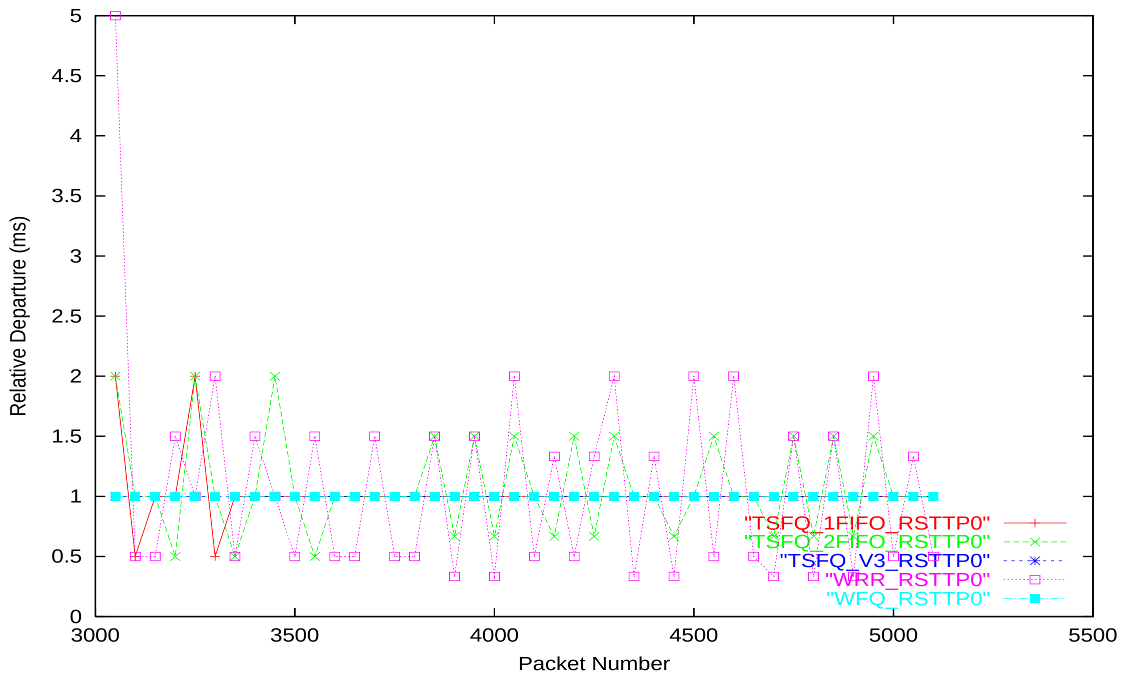

Figure 6.10: Relative Throughput - Multiple service levels, Fixed-size packets - 20 flows - Flow 1

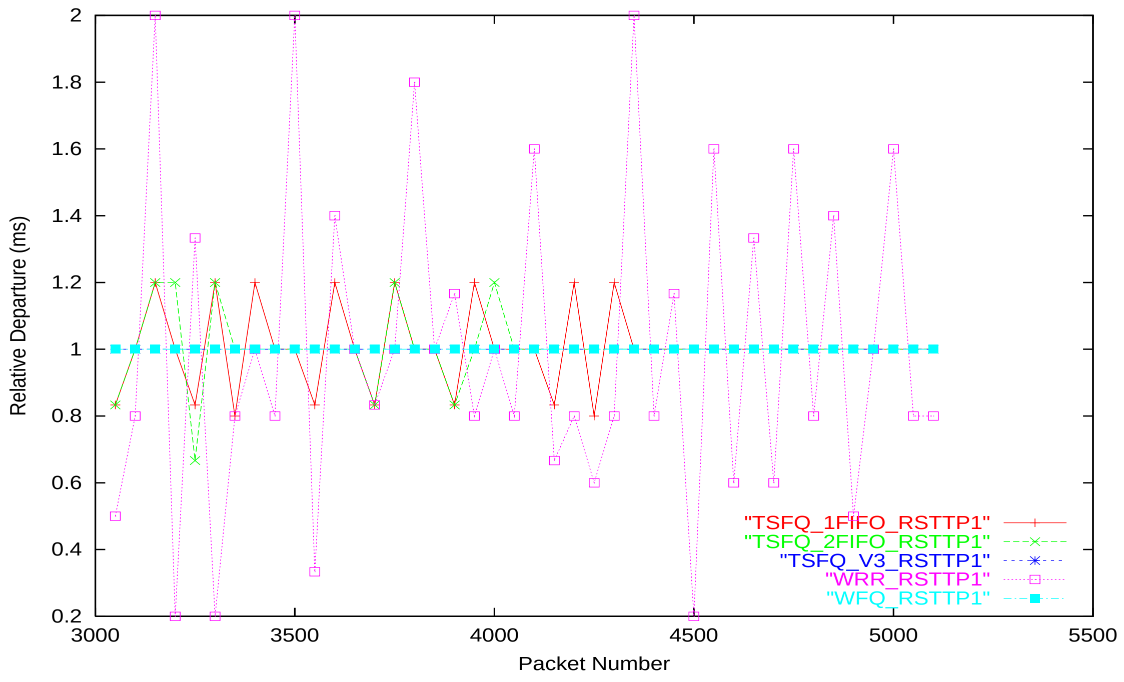

Figure 6.11: Relative Throughput - Multiple service levels, Fixed-size packets - 20 flows - Flow 2

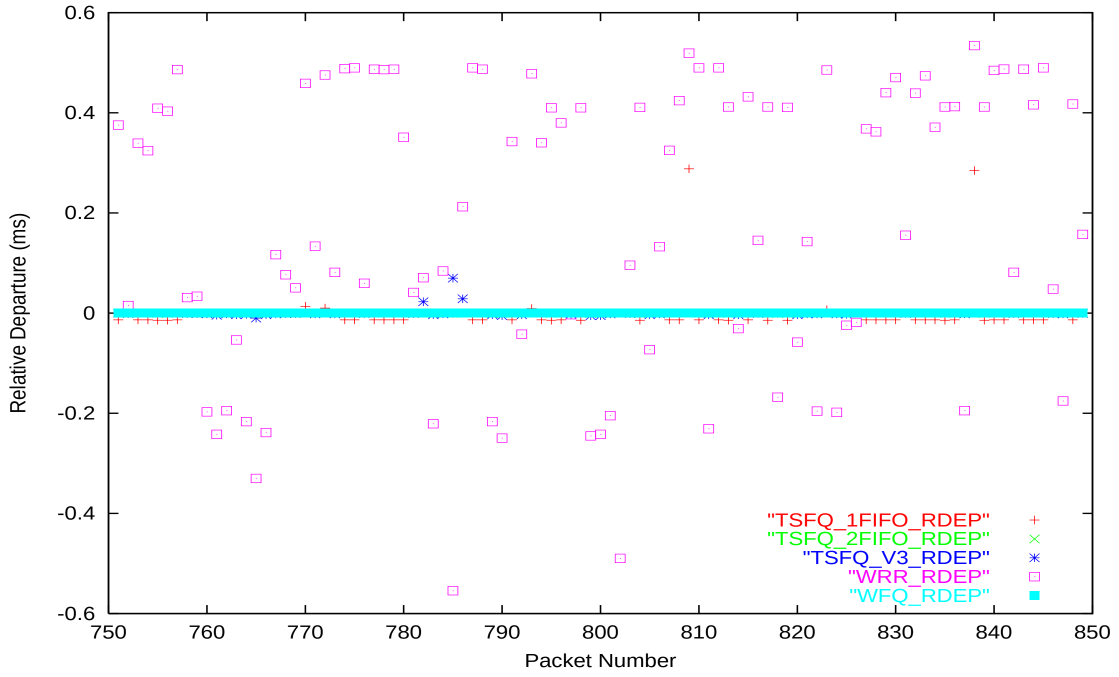

Figure 6.12: Relative Departure - Multiple service levels, Fixed-size packets - 1000 flows - over all flows

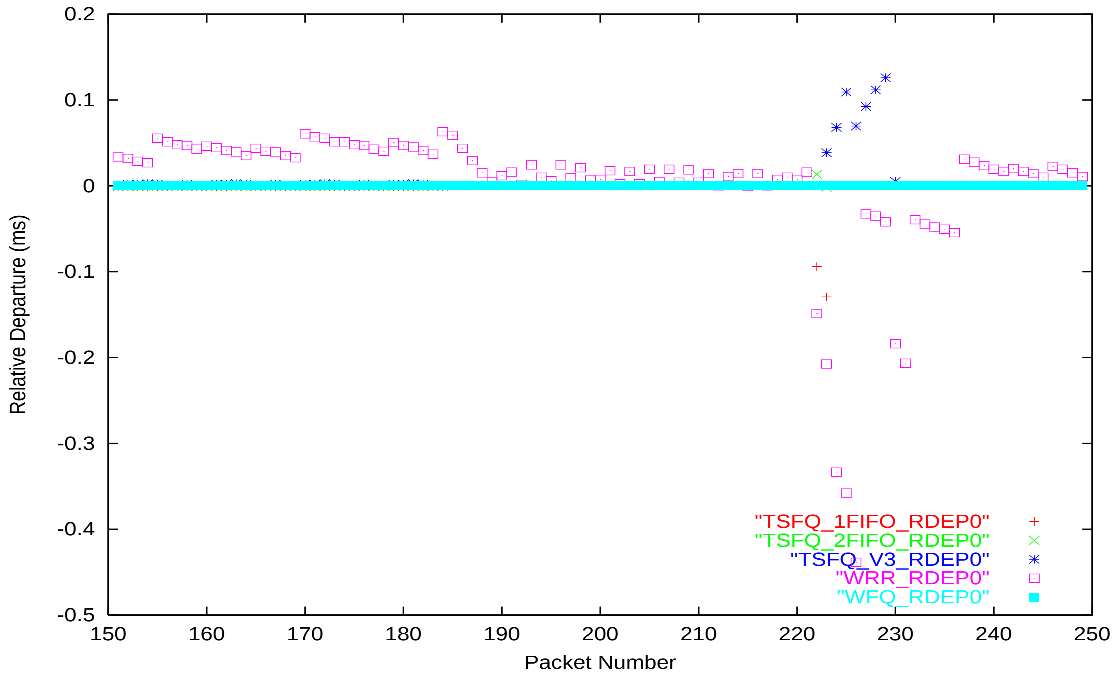

Figure 6.13: Relative Departure - Multiple service levels, Fixed-size packets - 1000 flows - over Flow 1

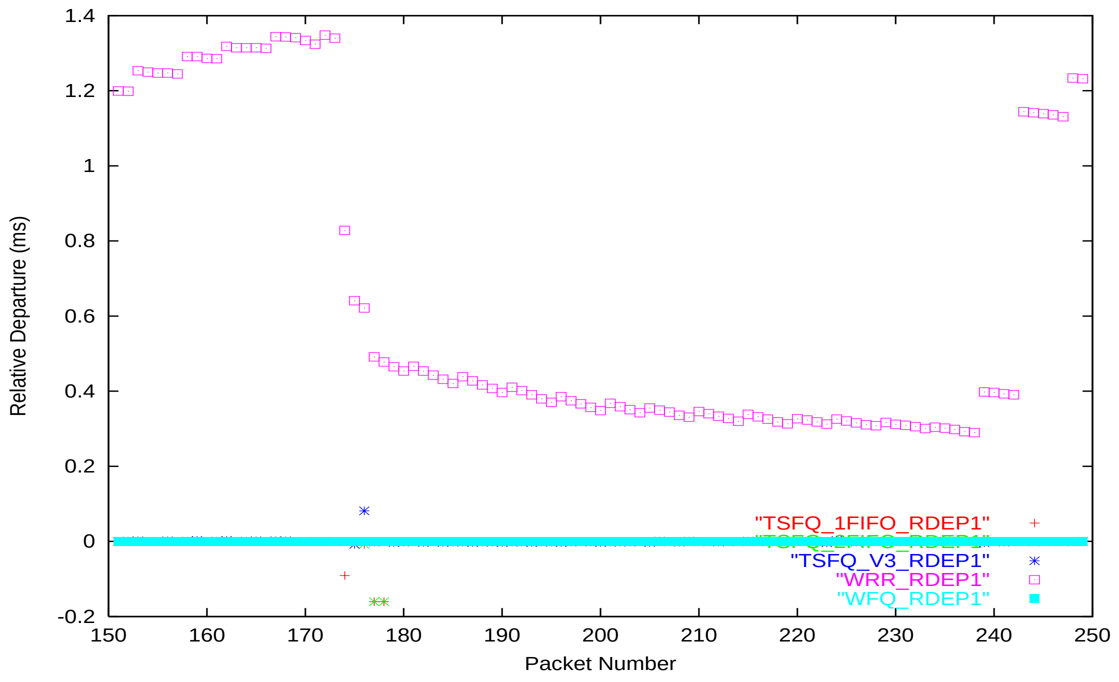

Figure 6.14: Relative Departure - Multiple service levels, Fixed-size packets - 1000 flows - over Flow 2

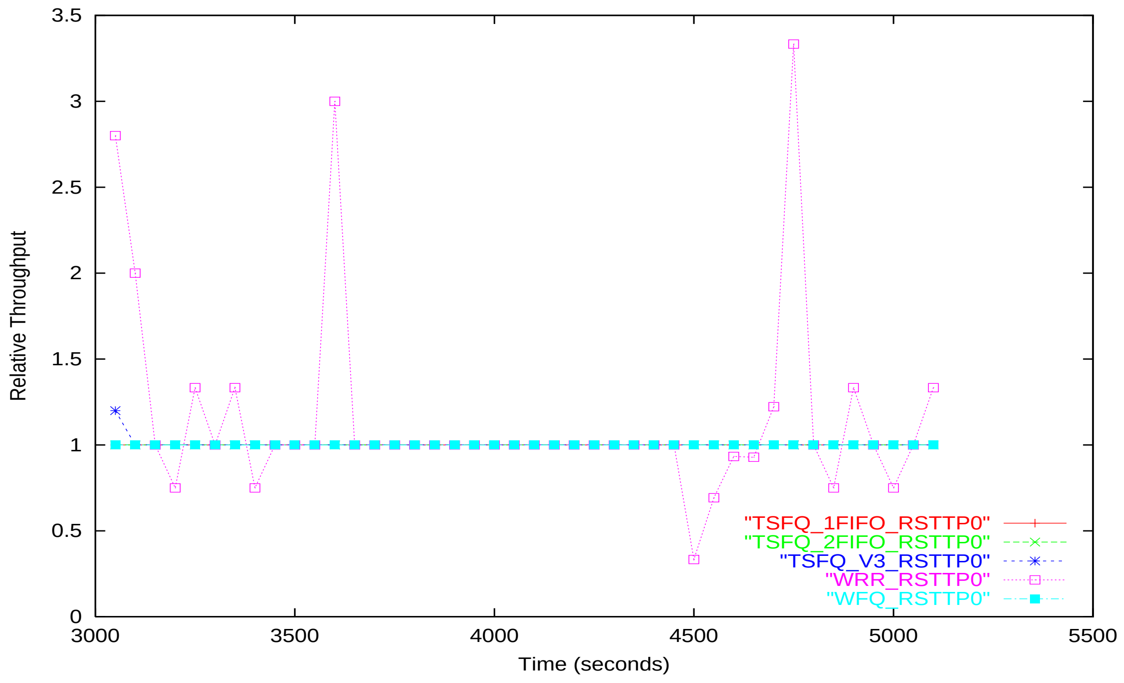

Figure 6.15: Relative Throughput - Multiple service levels, Fixed-size packets - 1000 flows - Flow 1

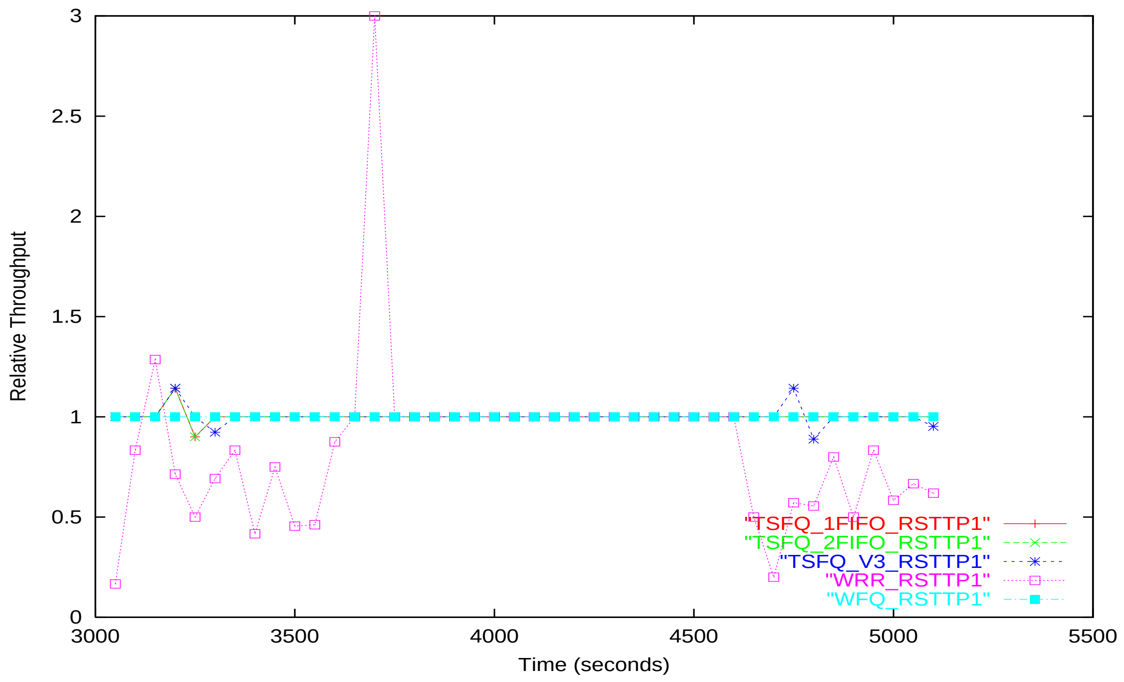

Figure 6.16: Relative Throughput - Multiple service levels, Fixed-size packets - 1000 flows - Flow 2

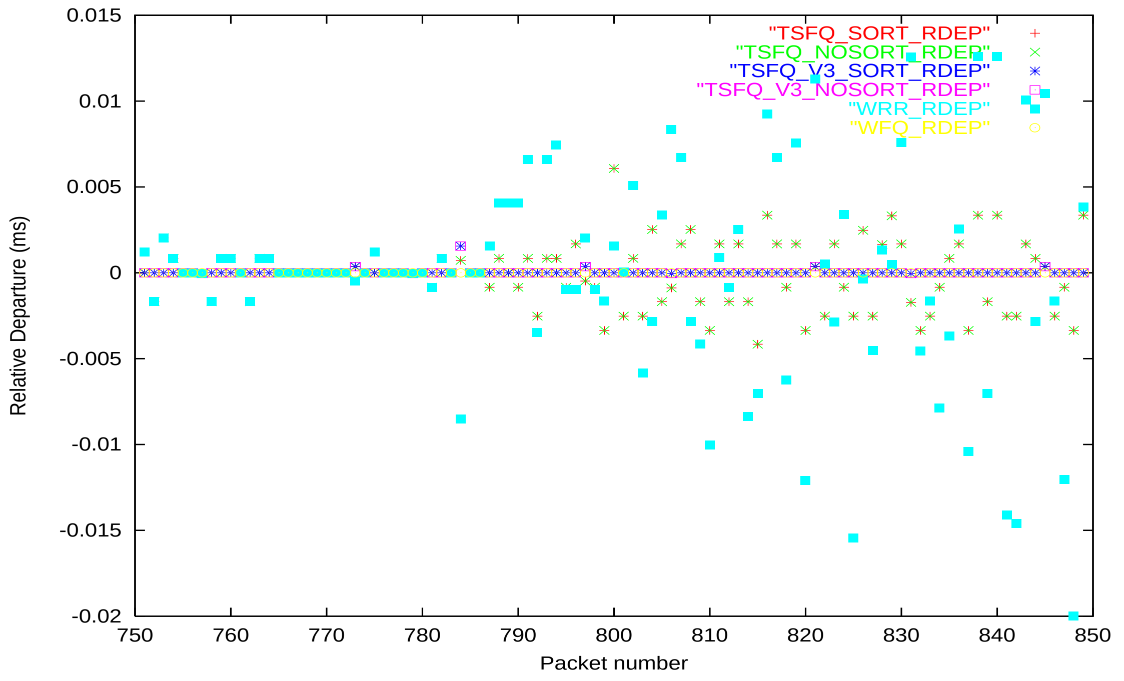

Figure 6.17: Relative Departure - Single service level, Variable-size packets - over all flows

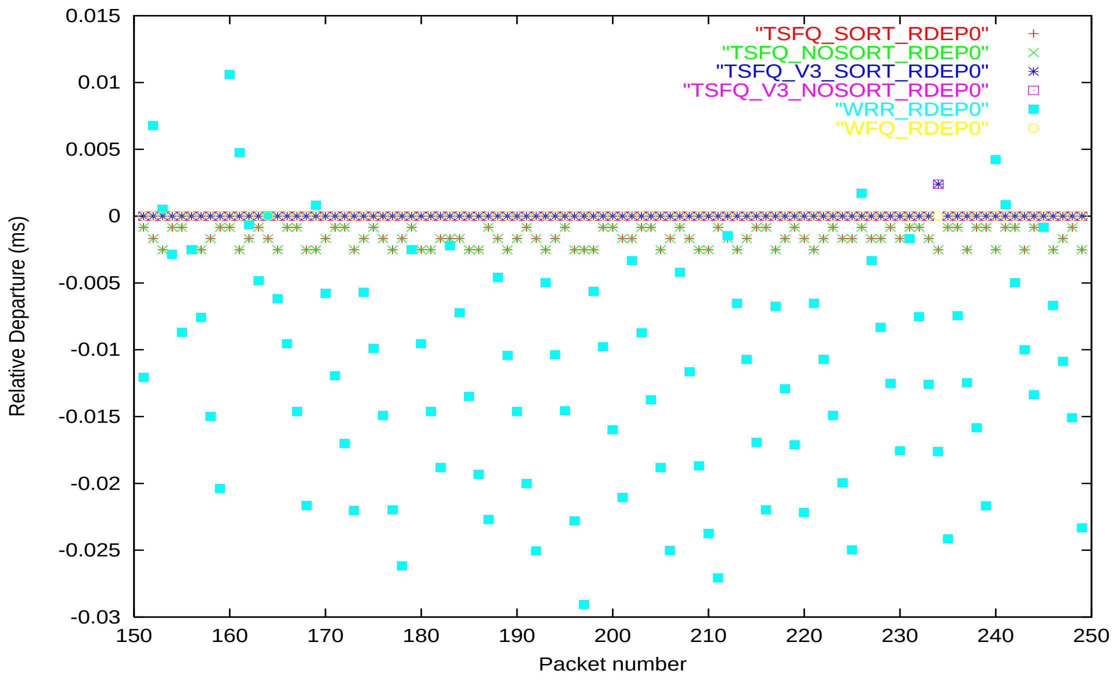

Figure 6.18: Relative Departure - Single service level, Variable-size packets - over Flow 1

in this experiment, the flows generated packets whose size varied from 100 to 600 bytes. Nearly 90% of the packets generated were of size 210 bytes and the remaining packets were of different sizes. In this experiment, we also compared the sorted versions of TSFQ with the unsorted versions. In the sorted version of TSFQ, the packets which are sent to the FIFO queues which hold packets of sizes in between the common packet sizes are sorted. In the unsorted version, these packets are simply added to the tail of the FIFO without any sorting.

Figure 6.17 shows the relative departure of packets from all flows and Figures 6.18 and 6.19 show the relative departure of packets from a single flow. Like the figures of fixed-size packets in the previous section, these figures also show that the TSFQ versions' departure order is closer to that of WFQ. We can also see from the figures that the unsorted versions of TSFQ have the same departure order as that of sorted versions. From this we infer that it is not really necessary to sort the packets of uncommon sizes since their arrival rate is low in a traffic with 90 - 10 common to uncommon sized packets ratio and the packets are spread over different FIFO queues.

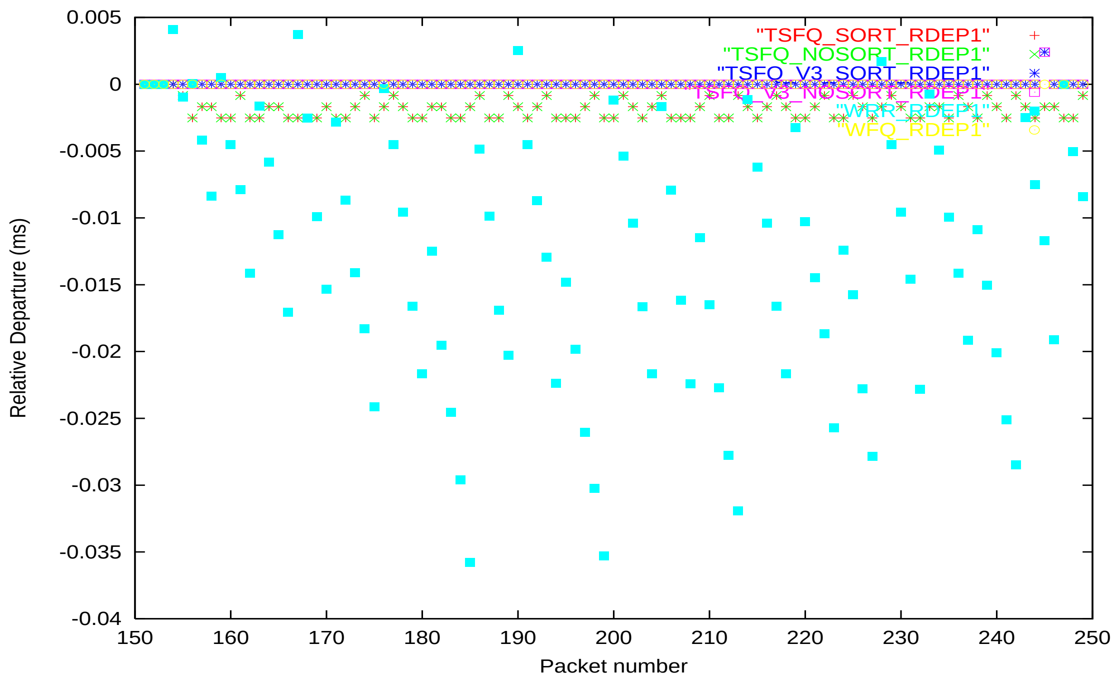

Figure 6.19: Relative Departure - Single service level, Variable-size packets - over Flow 2

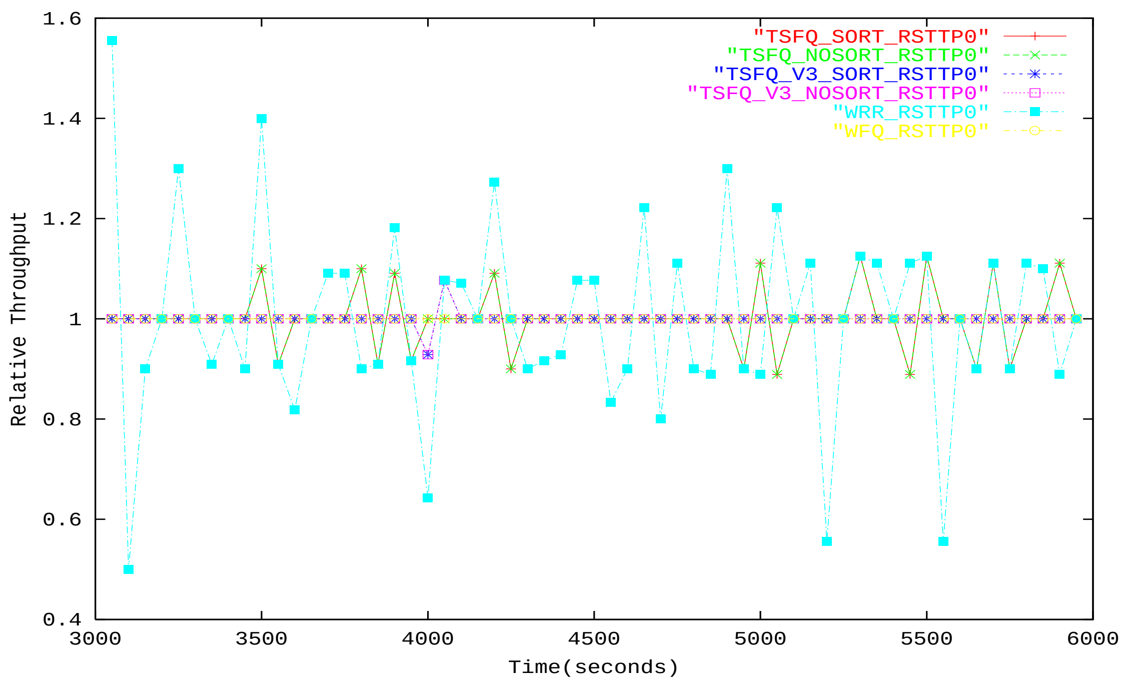

Figure 6.20: Relative Throughput - Single service level, Variable-size packets - 10 flows - Flow 1

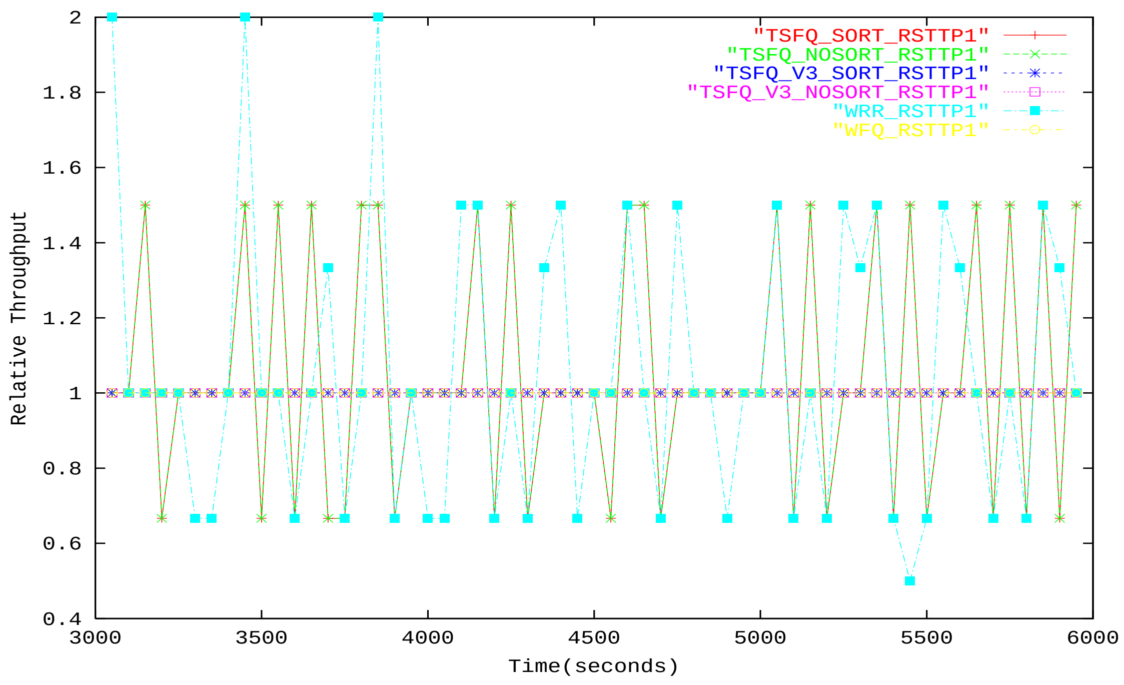

Figure 6.21: Relative Throughput - Single service level, Variable-size packets - 10 flows - Flow 2

Figures 6.20 and 6.21 show the relative throughputs of two flows under different schedulers. Similar to the case of fixed-size packets, WRR exhibits heavy fluctuations and hence poor fairness among flows. The figures also show that TSFQ versions, sorted and unsorted, provide better short-term fairness than WRR and TSFQ version 3 provide very similar fairness as that of WFQ. The unsorted version of TSFQ versions provide the same fairness as that of sorted versions of TSFQ.

6.4 Variable-Size Packets with Multiple Service Levels

In this experiment, we consider variable-size packet traffic, and we assume that different flows are assigned to different service levels. We simulated 1000 flows mapped to 10 different service levels, each with its own weight. The average rates of the flows varied from 1 kbps to 200 kbps. The flows generated packets of sizes varying from 100 bytes to 600 bytes. Similar to the experiment in previous section, the ratio of common to uncommon packet sizes was approximately 90 to 10. Figure 6.22 shows the relative departure of packets

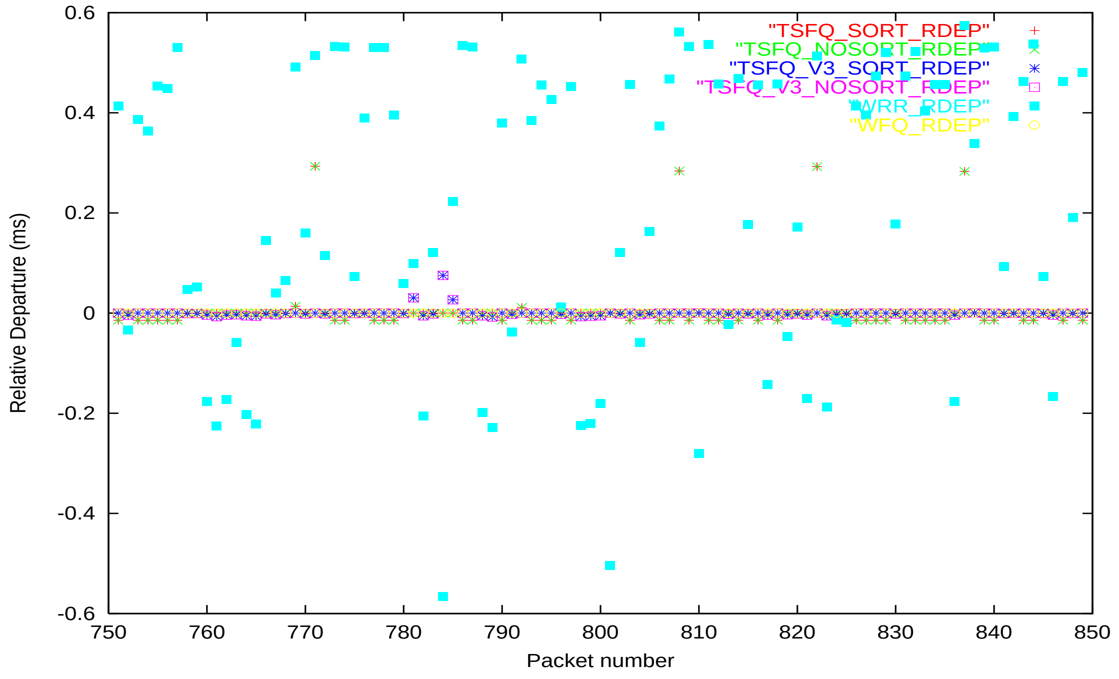

Figure 6.22: Relative Departure - Multiple service levels, Variable-size packets - over all flows

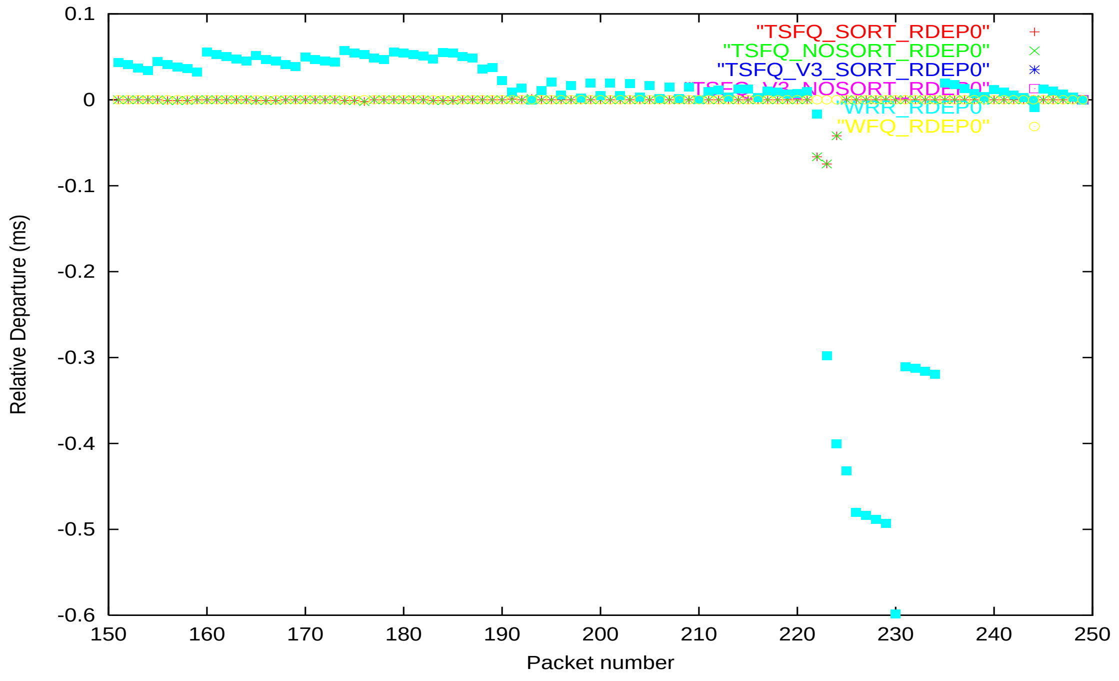

Figure 6.23: Relative Departure - Multiple service levels, Variable-size packets - over Flow 1

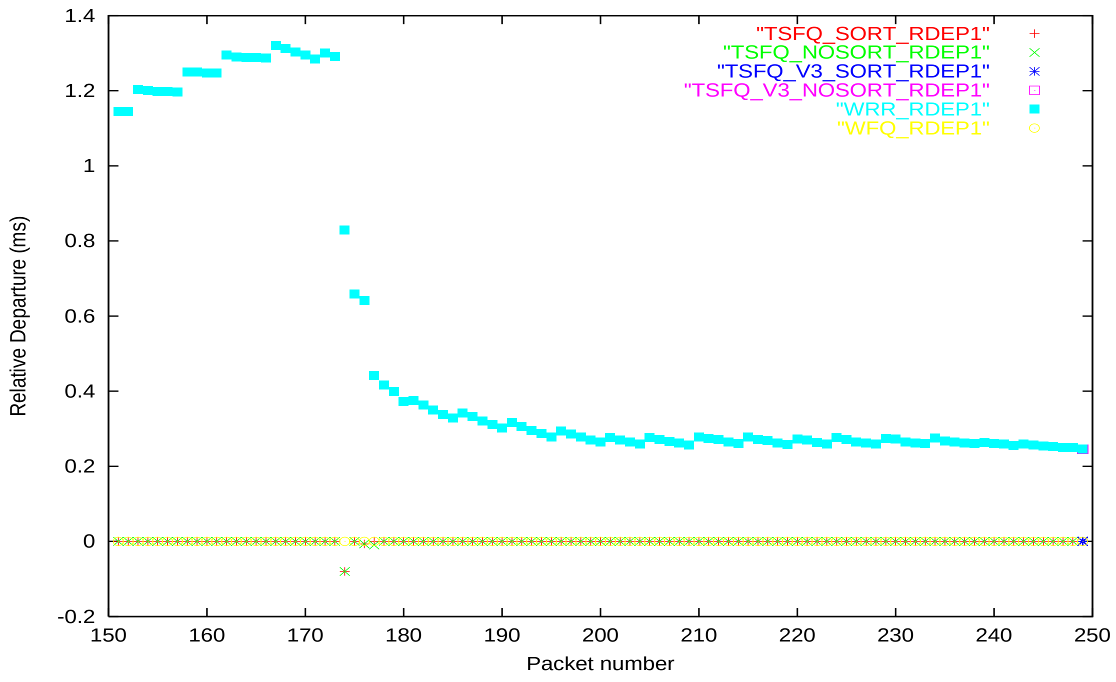

Figure 6.24: Relative Departure - Multiple service levels, Variable-size packets - over Flow 2

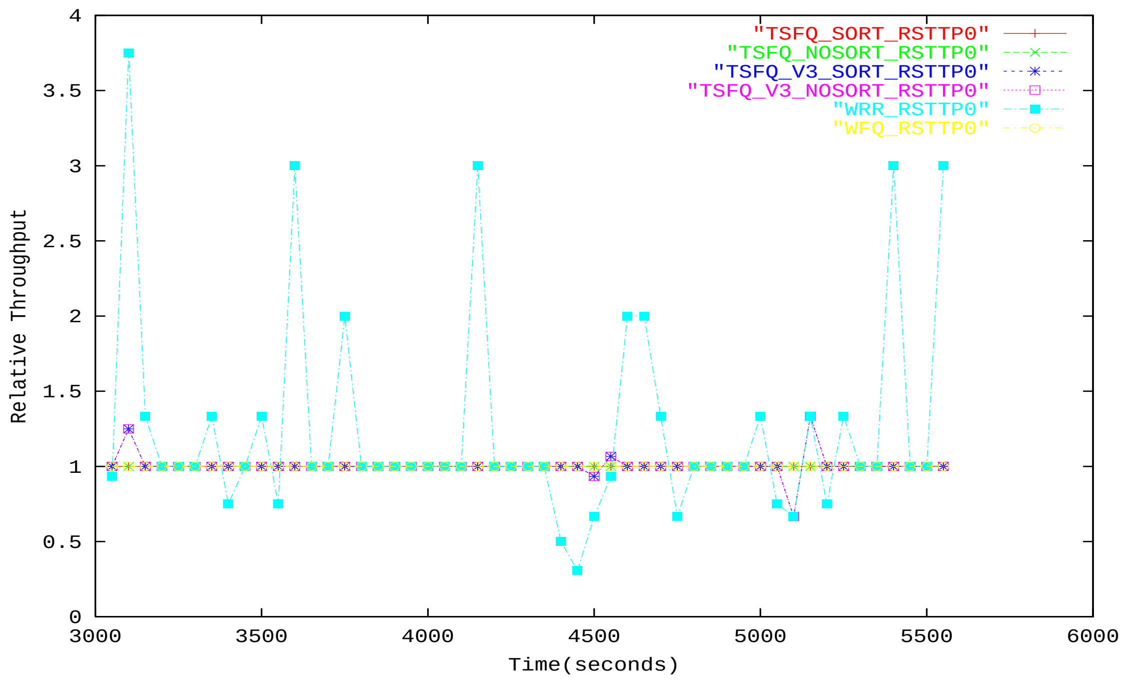

Figure 6.25: Relative Throughput - Multiple service levels, Variable-size packets - 1000 flows - Flow 1

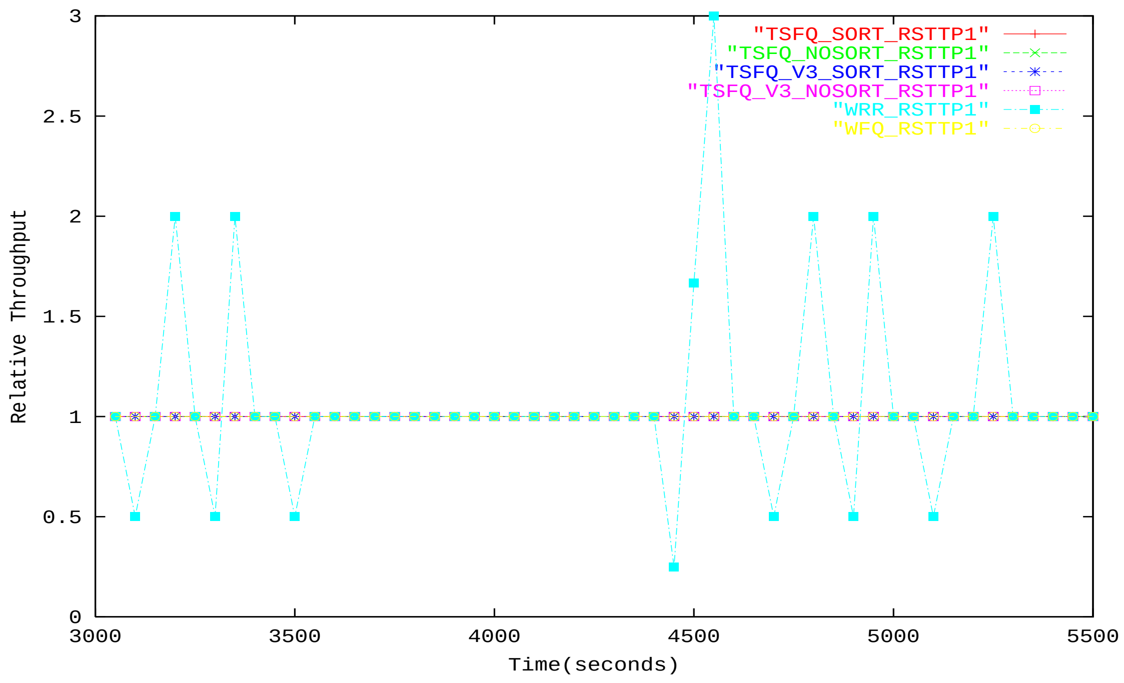

Figure 6.26: Relative Throughput - Multiple service levels, Variable-size packets - 1000 flows - Flow 2

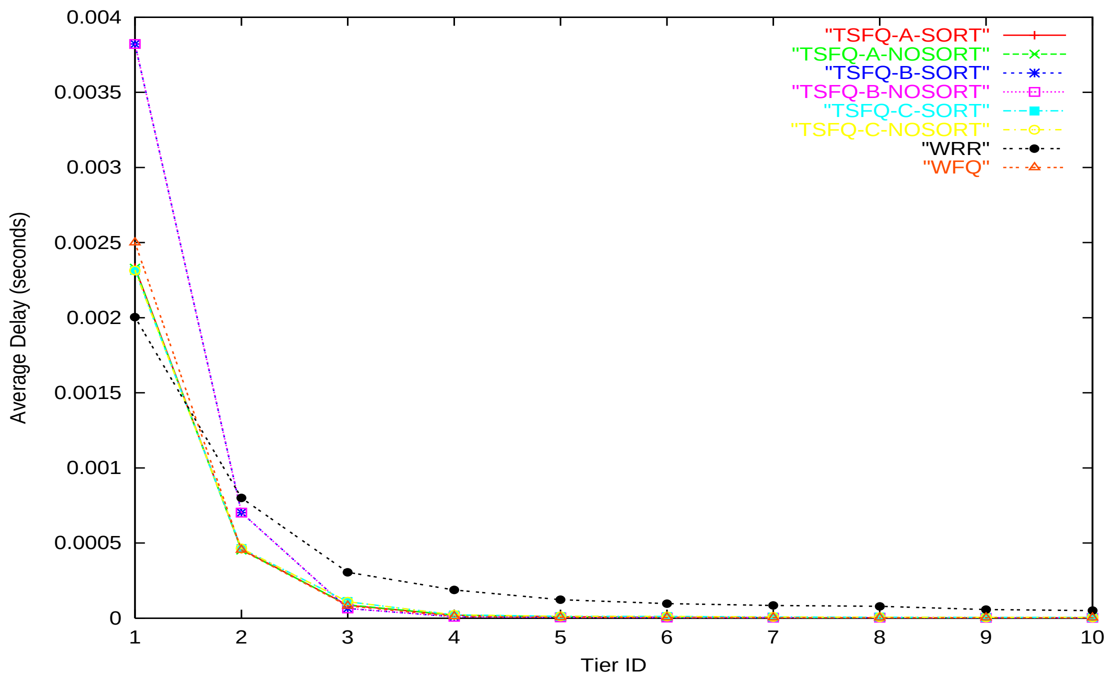

Figure 6.27: Average Delay - Multiple service levels, Variable-size packets - 1000 flows

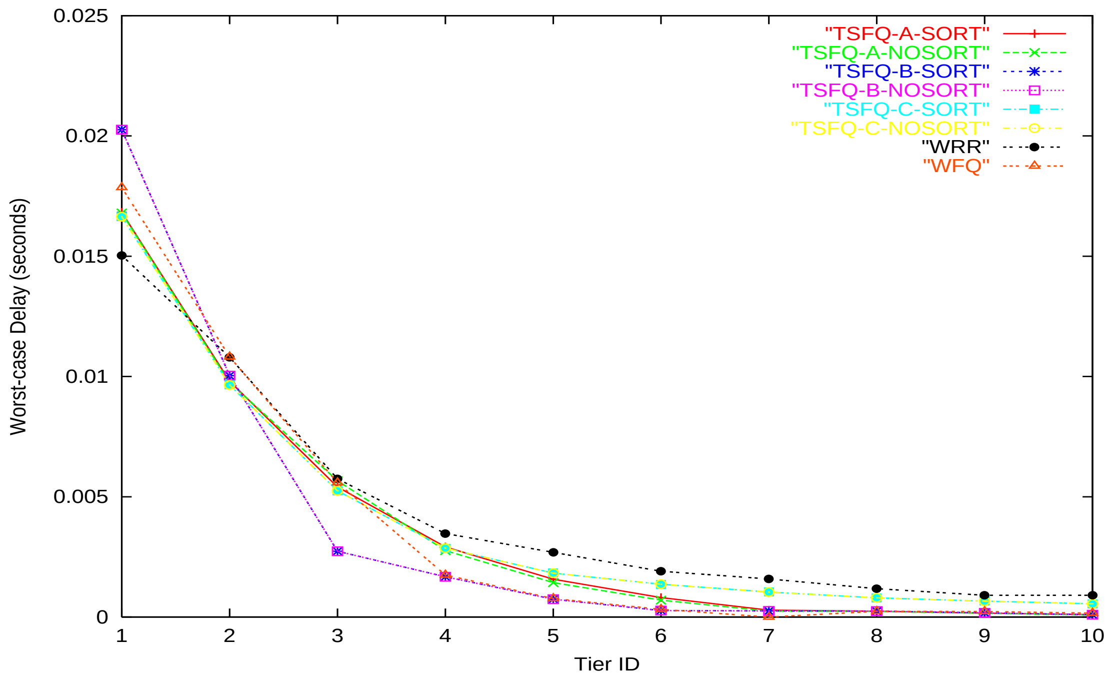

Figure 6.28: Worst Case Delay - Multiple service levels, Variable-size packets - 1000 flows

from all flows and Figures 6.23 and 6.24 show the relative departure of packets from a single flow. Figures 6.25 and 6.26 show the relative short-term throughputs of two flows under different scheduling schemes. The observations made from these figues are similar those from the previous section.

Figures 6.27 and 6.28 show the average and worst-case delays of the flows in different schedulers. The flows are gruoped by their tier ID. So, each point in the graph correspond to either the average or worst-case delay of all the packets belonging to a particular tier. We can observe from these figures that the delay values decrease with increase in the tier ID. This is because tiers with higher tier ID have greater weights. We can also see that the delay values of WRR differ widely from that of WFQ, whereas the packets served by the TSFQ schedulers experience delays similar to that of WFQ. Also, for all tiers except one, the worst case delay of WRR is the highest.

6.5 Discussion of Findings

The results obtained from the simulations can be summarized into the following points.

- Even though WRR is efficient in terms of running time complexity, it does not provide proper fairness among the flows served. In all the experiments, the short-term throughput of the flows exhibited heavy fluctuations, showing that the flows served by WRR may significantly over-perform or under-perform. Its service order also varies considerably from that of the PGPS and GPS service order.

- Since WFQ serves the packets in the GPS service order, it provides very good fairness among flows. But, WFQ has a running time complexity of O(log n), where n is the number of flows.

- TSFQ V.1 scheduler provides better fairness among the flows than WRR. Its packet service order is closer to that of the PGPS service order, even though there are few situations in which the service order changes from that of PGPS. We also know that TSFQ V.1 has constant time operational complexity.

- TSFQ V.2 scheduler performs similar to that of TSFQ V.1 scheduler in terms of fairness and service order, although the situations in which the service order changes differ from that of TSFQ V.1. Also, the changes in service order is slightly lesser than TSFQ V.1. This is achieved by doubling the number of service-level FIFOs, but still, TSFQ V.2 scheduler has a constant time operational complexity.

- TSFQ V.3 Scheduler peforms very similar to WFQ, in terms of fairness among flows and service order. There is a slight difference in the service order between TSFQ V.3 and WFQ under a few circumstances, since TSFQ V.3 serves packets within a service level in the order of their virtual start time rather than the virtual finish time. TSFQ V.3 scheduler also has a constant-time operation, even though it requires maintaining additional GPS queues.

- The experiments involving variable-size packets revealed that sorting FIFOs that hold packets of different sizes is not necessary for a traffic consisting 90% of packets with common sizes. Since Internet traffic consists of more than 90% of packets with common sizes [10], we can safely say that no sorting operations are necessary for packets with uncommon sizes.

Chapter 7

Summary and Future Work

7.1 Summary

We have considered the problem of applying the concept of traffic quantization in packet scheduling, in order to improve the efficiency without affecting the QoS guaranteed to the flows. Our objective was to develop a new packet scheduling scheme that provides good fairness among flows and at the same time, also operates at constant running time. We achieved this by assigning each flow in the network to one of a small set of service levels (tiers), in such a way that the QoS at least as good as requested by the flow is guaranteed. We presented three different versions of the new packet scheduler, TSFQ, each with its own advantages and issues. We showed how these schedulers operate in constant order of time, in both the fixed-size packet and variable-sized packet traffic. We also wanted to study the performance of the scheduling techniques in terms of fairness and service order. Finally, we showed from the simulation results that TSFQ versions offer better fairness than WRR, with TSFQ V.3 providing fairness as good as WFQ.

7.2 Future Work

Our work may be extended in many ways some of which we outline.

- We considered packet scheduling over a single link. Further research needs to be done to extend this packet scheduling scheme to operate over multiple links.

- We obtained results from running simulations on the implemention of the TSFQ versions in $ns-2$ simulator. It would be interesting to study the performance of these TSFQ schedulers from a linux kernel implementation.
- We studied the performance of TSFQ schedulers in terms of fairness and efficiency. It would be interesting to study the performance of the TSFQ schedulers in terms of network control, management and other costs.
- We selected WFQ and WRR to compare with TSFQ performance. It would also be interesting to compare the performance of TSFQ schedulers with other schedulers.

Bibliography

[1] http://www.cc.jyu.fi/ sayenko/src/wfq-1.2.4.tar.gz.

[2] Alok Aggarwal, Maria M.Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[3] Nikhil Baradwaj. Traffic quantization and its application to QoS routing. Master's thesis, NC State University, 2005.

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

[5] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Internetworking Research and Experience*, 1:3–26, 1990.

[6] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proc. of INFOCOM'94*, pages 636–646, June 1994.

[7] P. Goyal and H. M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. *IEEE/ACM Transactions on Networking*, 5(4):561–571, August 1997.

[8] S. L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13:462–475, 1965.

[9] Laura E. Jackson and George N. Rouskas. Optimal traffic quantization in packet-switched networks, 2004.

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

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

[12] M. Shreedhar and George Varghese. Efficient fair queuing using deficit round robin. *IEEE/ACM Transactions On Networking*, 4(3):375–385, June 1996.

[13] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. *ACM Transactions On Computer Systems*, 9(2):101–124, 1990.