MASTER'S THESIS

Issues in Resource Allocation and Design of Hybrid Gateways

by Ravichander Vaidyanathan
Advisor: John S. Baras

CSHCN M.S. 99-5
(ISR M.S. 99-8)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

Sponsored by : NASA

ABSTRACT

Title of Thesis:	ISSUES IN RESOURCE ALLOCATION AND DESIGN OF HYBRID GATEWAYS
Degree candidate:	Ravichander Vaidyanathan
Degree and year:	Master of Science, 1999
Thesis directed by:	Professor John S. Baras Department of Electrical and Computer Engineering

Considerable attention has been focussed on active queue management and fair resource allocation techniques in the Internet. However, few real-world instances exist of deployment of IP routers/gateways which implement such techniques. In the first part of this thesis, we analyze the implementation feasibility and overhead of these schemes to determine whether this overhead represents an obstacle to deployment. To this end, we employ a novel approach with real traffic traces from the Internet and a passive gateway simulator. Having established the feasibility of such algorithms, we turn our attention to buffer management techniques in the presence of fair resource allocation. Our specific focus is on developing an effective buffer management technique for satellite networks. The limitations of existing schemes lead us to propose a new buffer management

scheme designed with our problem space in mind. Finally, we look at a class of satellite enhanced gateways, termed as connection splitting or spoofing gateways, proposed for implementing high performance satellite systems. The specific design issues peculiar to this class of gateways are analyzed. A novel architecture for fair resource allocation in spoofing gateways is then proposed. The efficacy of our architecture in providing fairness and protecting adaptive flows against misbehaved and non-adaptive flows is demonstrated by means of simulation.

ISSUES IN RESOURCE ALLOCATION AND DESIGN OF HYBRID GATEWAYS

by

Ravichander Vaidyanathan

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
1999

Advisory Committee:

Professor John S. Baras, Chair
Professor Leandros Tassiulas
Dr. M. Scott Corson

DEDICATION

For Dad and Mom

ACKNOWLEDGMENTS

I am grateful to my advisor, Dr. John S. Baras, for his encouragement and guidance throughout my time here. This work would not have been possible without his support. I am indebted to the research environment at the Center for Satellite and Hybrid Communication Networks (CSHCN) and the Institute for Systems Research (ISR) for helping shape my Masters' career. I would also like to express my appreciation for the support received by several sponsors throughout my studies at the University of Maryland. This work was supported in part by NASA under the Commercial Space Center program, cooperative agreement NASA-NCC3528, and by Lockheed Martin Global Telecommunications through a contract. I have also benefited from an exciting summer CSHCN internship at Bellcore (now Telcordia Technologies).

This thesis would have looked very different if not for the "guys" at the Center and ISR who were always there when I needed technical or moral support. They made it that much more fun to stay up all night working: to Manish, Mingyan, Roshni and Vijay. Many thanks to Steve, for showing me the "way of the graduating student". To Shashi and Himanshu, for making life 10,000 miles away from home more livable. To my rejuvenating escapes to NY; to party with my Radha akka. She made it seem like there was always family in driving range.

And finally to Preethi, my fiance, for everything.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The Internet, a heterogeneous interconnection of networks that span all corners of the world, has come to govern much of modern communication. The rapid interconnection and interoperation between a score of technologies that comprise the Internet has been made possible by the philosophies underlying the design of the protocol that glues most of the Internet together, the Internet Protocol (IP) [1].

The essential philosophy behind the design of IP was to keep it simple and robust. IP offers a best effort, datagram type of service interface and maintains minimal state in the network. The service offered by IP is connectionless and operates on a hop by hop basis, with each intermediate node along a route being aware of only its next hop neighbor. In such a scenario, applications requiring value-added services such as reliable delivery utilize suitable end-to-end protocols in the Internet Protocol suite such as the Transmission Control Protocol (TCP) [37].

The exponential growth in user demand for bandwidth and the increasing interest in interactive and multimedia applications led to the development of more sophisticated network technologies. In the forefront were technologies such as Asynchronous Transfer Mode (ATM) [2] which offers benefits such as Quality of Service(QoS) differentiation in traffic traversing the network and bandwidth on demand at the price of additional complexity at the core of the network. Such technologies are aimed at providing differentiated service levels to incoming traffic based on requirements such as delay, loss and jitter. The best effort paradigm is replaced by a paradigm which attempts to shape and optimize the behavior of incoming traffic. This is accomplished by policy based resource allocation in the intermediate nodes. Specifically, the shared resources we refer to are the processor and the available buffer space at the intermediate node.

While ATM itself has not found the kind of widespread deployment that it appeared to be destined for, it brought into harsh relief the limitations of the best effort paradigm. Ongoing efforts such as the Differentiated Services Charter of the IETF [3] are focussed at defining coarse differentiated levels of service for Internet traffic. Coupled with the increasing scarcity of bandwidth, this has led to widespread interest in the optimization of traffic behavior.

In the best effort paradigm, no effort is made by intermediate nodes to optimize traffic behavior and resource allocation at the intermediate nodes is often not explicitly managed. In such a case, the dynamics of traffic are governed to a large extent by end-to-end protocols such as TCP. Recent research [36] has

shown the undesirable effects of such a lack of policy in resource allocation such as unfair link sharing and reduced overall performance. In the absence of policy based resource allocation in the intermediate nodes, malicious users can procure an unfair share of the resources.

Among the fundamental requirements for traffic optimization and differentiated levels of service at intermediate nodes are good algorithms for buffer management and scheduling. In spite of extensive research in this area, there are few instances of real-world deployment of routers and gateways with such fair resource allocation techniques [13]. The lack of such deployment leads us to investigate the implementation feasibility of fair resource allocation techniques and whether the overheads involved are an obstacle to deployment.

Having analyzed the feasibility of such resource allocation techniques, we turn our attention to an analysis of the algorithms for resource sharing. A considerable base of research already exists in this area. Our work is delineated by our specific focus on satellite networks and their interaction with terrestrial networks.

Satellites have always played a crucial role in Internet connectivity. The linking of remote regions of the world and third world countries which lack a good communication infrastructure to the Global Information Infrastructure (GII) is best accomplished by satellite networks. The explosion in portable devices and the requirement that they be connected to the Internet afford further roles for satellite interconnectivity.

The physical characteristics of satellite networks create several unique

problems in networking. Among the best studied are the nature of errors on satellite links and the long latency over satellite. Satellite links are less reliable by several orders of magnitude than terrestrial links in common use today. Errors on a satellite link also tend to be bursty in comparison to terrestrial links. Most protocols in use today were specifically developed for terrestrial networks and perform poorly over satellite links. Concerns such as interoperability surface in the development of satellite-specific protocols. Hence, the focus of current research is to selectively optimize existing protocols over satellite links.

Well known solutions to improve the reliability of satellite links such as strong Forward Error Correction (FEC) focus on encoding for error correction at the link layer. A survey of common techniques including protocols such as the SNOOP protocol, which performs link-level Automatic Repeat Request(ARQ) by using TCP acknowledgements, can be found in [46]. Commercial products such as COMSAT Corporation's ATM Link Enhancer (ALE) [4] utilize cell interleaving and Reed Solomon encoding to enhance link characteristics. Our focus is on the performance and optimization of higher layer protocols, specifically IP and TCP, of the Internet protocol suite. Much of our attention is focussed on improving the end-to-end performance of TCP, the most commonly used protocol for reliable data transport in the Internet.

We term intermediate nodes, which serve as interconnects between satellite and terrestrial networks, *hybrid gateways*. Such gateways, incorporating innovative modifications to the TCP protocol, were first introduced in the work

that led to the development of the Turbo^{TM} Internet product of DirecPC^{TM} [48] and the high data rate satellite gateway developed in [44]. Throughout this work, our focus is on solving the resource allocation and design problems particular to this class of gateways for enhanced performance over satellite. The rest of this dissertation is organized as follows.

Chapter 2 evaluates the implementation and deployment feasibility of resource allocation techniques in gateways and routers in the Internet without specific reference to satellite. To this end, we employ a novel approach utilizing Internet traffic traces and a passive gateway simulator. The overhead of various fair scheduling disciplines are quantified and techniques to reduce this overhead are explored. The results of chapter 2 demonstrate both the need and feasibility of fair resource allocation techniques.

Having demonstrated the practical feasibility of resource allocation techniques, we turn our attention to commonly employed buffer management techniques in the presence of fair resource allocation. Their limitations when deployed on hybrid gateways lead us to propose a new buffer management technique designed with the unique characteristics of satellite networks in mind. We analyze and evaluate our buffer management technique by simulation and compare it with several commonly used buffer management techniques. Our results suggest that the proposed technique is well suited for the domain of hybrid gateways. This forms the subject of chapter 3.

Any work on hybrid gateways is incomplete without a discussion of a specific

class of proxy based solutions proposed for implementing high performance satellite systems first introduced in [48]. A proxy based architecture is deployed at the hybrid gateways to improve the observed end-to-end performance over satellite networks. We examine such an architecture in chapter 4 and propose a new architecture for resource allocation in such a setup. We demonstrate that the proposed architecture results in reduced buffering with no loss of throughput performance and is effective in protecting fragile adaptive flows threading the hybrid gateway.

Finally, we conclude with some observations on the design and resource allocation issues involved in the implementation and deployment of hybrid gateways. The contributions and importance of this work are delineated.

Chapter 2

A Feasibility Analysis of Fair Queueing

2.1 Introduction

Most gateways and routers deployed in the current internet deploy a single queue in which packets awaiting service are enqueued. The queues are First In First Out (FIFO) queues and usually drop the last packet in the queue, in the case of queue overflow. Such systems do not seek to optimize the behavior of traffic on the Internet. Traffic shaping is left to end system implementations.

The need for optimization of traffic behavior is motivated by the varying requirements of different types of traffic on end to end parameters such as delay, jitter, bandwidth and reliability. From a service point of view, traffic may indeed be classified by its service requirements.

A fundamental requirement for treating traffic "fairly"(we define fairness formally later) or preferentially is the separation of the traffic into classes or flows. In the terms of the Internet protocol, a flow may be loosely defined as a

sequence of IP datagrams which traverse similar paths through the network. While a flow is end-to-end, we confine our view of a flow by observation at an intermediate node, such as a gateway. The view of a flow at an intermediate node, such as a queueing system is termed in the literature as a *conversation* [5]. We use the term flow to imply a conversation in the rest of this discussion.

Scheduling disciplines and buffer management policies together determine the fraction of service provided (or resources allocated) by an intermediate node to a particular flow. Thus they determine the traffic shaping behavior of an intermediate node in the network.

Scheduling disciplines may be broadly classified as *work conserving* and *non-work conserving* [13]. Work conserving schedulers refer to the class of schedulers which are idle only if there is no packet in the system currently awaiting service. Non-work conserving schedulers may be idle even if packets are enqueued in the system. Non-work conserving schedulers find applications in traffic shaping and the provision of delay jitter guarantees. However, with the rapidly decreasing cost of memory delay, jitter guarantees are often satisfied by buffering at the endsystem. The feasibility of non-work conserving schedulers for deployment in a real system is questionable and we shall not discuss them further.

Priority scheduling refers to the specification of a number of priority levels for different types of traffic. Priority enables a scheduler to assign differentiated service levels to incoming traffic e.g. the provision of lower mean queueing delays to real-time applications. Often priority scheduling is associated with some form

of resource reservation, the details of which are beyond the scope of this discussion.

A slightly orthogonal but highly relevant issue to the implementation of a scheduling discipline is the level of aggregation or the flow definition. Flows may be more strictly defined by specifying classification mechanisms. Typical flow classification mechanisms operate on the IP header and define a flow as a source-destination IP address pair. We define the *granularity* of a flow in relation to the number of distinct transport layer flows it may encompass at any given instant of time. The granularity of a flow definition may be coarse, e.g. source-destination IP subnet number. An instance of a finely granulated flow definition is source-destination IP address pair + protocol + source-destination transport protocol port number. The level of aggregation is thus tied into the granularity of the flow classifier.

The definition of fairness is tied closely into the system requirements and the levels of service that are required to be supported. A scheduler for a best effort traffic flow can arguably trade implementation efficiency for looser fairness and delay bounds. A scheduler in a guaranteed service model would ideally have tighter bounds on the above. Another design issue is related to scheduling between flows of the same priority class in a priority scheduler.

We briefly review existing schemes for fair queueing in the literature. The motivation is not to furnish an exhaustive review but to set up an evaluation of the complexity and performance tradeoffs between a subset of these schemes.

The evaluation methodology is easily applicable to many scheduling algorithms not explicitly discussed in this survey and the evaluation process follows along the same lines. We ignore details such as the assignment of prioritized traffic levels and focus to a large extent on the best-effort type of traffic. Our objective is to setup a framework for evaluating the overhead incurred in the implementation of various scheduling algorithms.

2.2 Scheduling Algorithms

We cite a number of parameters from the literature to evaluate and define the performance of a fair queueing algorithm. We use these parameters to contrast the various schemes discussed in this section.

1. *Isolation and protection :* the insensitivity of a flow to the (mis) behavior of other flows in the system.

2. *Fairness and Bandwidth Sharing :* we refer to the concept of the max-min criterion. In a system where low rate flows are competing (fairly) with higher rate flows, the low rate flows must be serviced at their maximum request. The remaining bandwidth is then divided up between the higher rate flows.

3. *Delay and loss bounds :* This refers to the provision of bounded delays and loss guarantees to flows irrespective of the behavior of other flows in the

system. Such bounds are usually available at the expense of a resource reservation protocol.

4. *Implementation efficiency :* The ease of implementation of the scheduling algorithm. Throughout this chapter we use the so-called big-O notation to quantify the scalability of specific algorithms, i.e. their growth in terms of computational complexity.

Nagle in [5] noted that the general problems of congestion in datagram networks are not solved by the availability of infinite buffer space at the gateway. The author observed that FIFO queueing allocates the most resources to the source which sends the maximum data. A misbehaving sender, such as a voice application which sends at a constant rate, could capture more resources than a well behaved transport protocol implementation which is adaptive and reacts to loss. Nagle's proposal replaces the single FIFO queue at the gateway by multiple queues, one for each source host. The multiple queues are then serviced in a round robin fashion, with empty queues being skipped over. In the case of buffer overflow, a packet is dropped from the longest queue.

The intent of offering a fair share of the available bandwidth in [5] is defeated if different sources have different packet lengths. Also, in terms of service of a packet based on its arrival time, static round robin service does not provide continuity. The obvious flaws in Nagle's scheme and its discrimination against sources with smaller packet lengths led to the proposal of Generalized Processor

Sharing (GPS) [6] or bit-wise round robin. GPS is based on the fluid flow model where packets are assumed to be infinitesimally divisible. Multiple flows can traverse the same link simultaneously, at differing rates. GPS is an ideal scheduling discipline but is practically infeasible.

The packet approximation to GPS, Weighted Fair Queueing(WFQ) [7], functions by simulating the fluid flow model in parallel with the packet by packet approximation to identify backlogged flows. WFQ works by calculating the time at which a packet *would* have left the system in an ideal GPS environment, the so-called timestamp. Packets are timestamped with their departure times and then served in the order of their timestamps (finish numbers). A buffer management policy is also outlined in [7], where in the case of an arriving packet finding all buffers full, packets are discarded in the order of decreasing finish numbers until there is enough room for the arriving packet. WFQ provides good delay and fairness bounds but is prohibitive in terms of computational complexity.

[18] studies the components of an FQ server and discusses implementation issues for WFQ. An evaluation of the efficacy of algorithms and associated data structures is studied with the aid of a network simulator and a real workload. The simulation of a bit-by-bit round robin server to compute the timestamps in WFQ may require the processing of O(N) events per packet processing time. This complexity involved in computing timestamps makes the scheduler difficult to deploy in terms of implementation cost. Also, even the cheapest implementation

of the scheduling discipline requires the sorted insertion of a packet based on the timestamps. The worst case running time of efficient algorithms for this problem space [19] scales as $O(log_2 N)$ where N is the number of active flows. We note that the second problem is one that is shared by any fair queueing algorithm that uses some form of packet reordering or tagging and is not unique to WFQ.

Stochastic Fair Queueing [16] attempts to solve the flow classification problem and eliminate the overhead associated with it. A hash function is implemented in Stochastic FQ to assign a queue to an arriving packet. In the case of a collision, packets from multiple flows share the same queue. In order to avoid persistent unfairness to colliding flows, the hash function is regularly perturbed. The term stochastic arises from the random fairness received by each flow, as a function of the hash scheme. A *buffer theft* mechanism is also outlined where an arriving packet that finds all the buffers full steals a buffer slot from the longest queue. An efficient implementation for buffer theft is also discussed in [16]. The techniques described in Stochastic FQ can be applied to most fair queueing algorithms without loss of generality. At the altar of implementation efficiency, the delay and fairness bounds of WFQ are sacrificed in this scheme.

Deficit Round Robin(DRR) [17] attempts to solve another implementation problem associated with fair queueing: the overhead involved in choosing a packet to schedule on the outgoing link. DRR builds on Weighted Round Robin (WRR). WRR enhances round robin service by assigning weights to the connections based on their mean packet lengths. The scheme assumes that such a

mean value is readily available or can be estimated. DRR takes the concept further by eliminating the need for a mean value. In DRR, each connection is assigned a quantum of service and a deficit counter. In each round, the scheduler attempts to serve a quantum's worth of packets from each queue. If the packet at the head of the queue exceeds the quantum of service, then the quantum is added to the deficit counter and the queue is skipped over. Empty queues have their deficit counter reset. These schemes suffer from short term unfairness since in time intervals less than that of an entire round of service, they may be unfair. For a large number of connections this time interval may be quite large.

Schemes such as DRR and Stochastic FQ were designed with an implementation perspective in mind and differ from the more rigorous approach embodied in WFQ and its variants, which we describe subsequently.

An approximation to WFQ, Self Clocked Fair Queueing(SCFQ) [8] speeds up the biggest limitation of WFQ, the timestamping. SCFQ computes timestamps based only on the packet in service, thus scaling as O(1) in computational complexity. The performance improvement of SCFQ comes at the cost of loss of the delay and short term fairness bounds of WFQ. The delay bounds of SCFQ grow linearly with the number of active flows.

Start Time Fair Queueing (SFQ) [9] leverages the computational benefits of SCFQ but demonstrates better bounds on the worst case delay and the short term unfairness. SFQ schedulers maintain a virtual time and service packets based on their start tags. The start tag of a packet is assigned based on the

system virtual time and the finish time of the packet ahead of it in this flow. SFQ thus utilizes the notion of both start and finish tags. [9] also notes that, under some conditions, SFQ has lower end-to-end delay bounds than WFQ. Intuitively, this can be attributed to the fact that by using the start tags to schedule service, SFQ schedules a packet as early as possible while WFQ delays packet service as long as possible by using the finish numbers.

Several scheduling disciplines such as VirtualClock [10] and Delay Earliest Due Date [11] to name only a few are not described here. We note that in terms of design and implementation the overheads suffered by these disciplines and their performance is comparable. Our evaluation revolves around a feasibility study conducted by analyzing the overhead of these scheduling disciplines with a trace simulator driven by real traffic traces from the Internet. We first define the overhead of a fair queueing algorithm in more detail.

2.3 A Framework for Complexity and Overhead Evaluation

We now present a framework within which to evaluate the feasibility of fair queueing, with respect to implementation and deployment expense. Earlier investigations [18] [16] [17] concerned themselves with efficient algorithms to minimize the overhead incurred in the deployment of fair queueing schemes. We attempt a different approach that is based on the following :

1. A quantification of the different overheads involved in fair queueing and their application to various fair queueing algorithms discussed previously.

2. An analysis of live traffic traces with the help of our trace simulator, A Simple Queueing Gateway (ASQG). This includes

   (a) A study of the effects of flow definition on the scalability of fair queueing schemes.

   (b) A quantification of the operating requirements of an intermediate node for scalable fair queueing deployments at different areas in the Internet.

   (c) A performance comparison between different scheduling algorithms in terms of implementation efficiency.

We also note that studies of flow classification have been performed in [21] [22]. These, however, are relevant to IP over ATM or IP switching scenarios and tackle a different problem space. We believe that our analysis framework is extensible to most fair queueing schemes and furnish specific instances of fair queueing algorithms within our framework.

The following section quantifies the fair queueing overhead formally and contrasts the overheads incurred by different scheduling algorithms. Efficient implementation techniques for various aspects of a fair queueing algorithm are also presented. We then describe our simulation study of fair queueing along with an architectural overview to enable a deployment perspective.

2.4 Quantification of Fair Queueing Overhead

The following overheads may be identified as directly or indirectly resulting from a fair queueing deployment.

1. *Flow Classification :* Identification and separation of flows into multiple queues. This is a fundamental requirement of any form of fair queueing algorithm, but is rarely lumped together with some of the other considerations.

2. *Scheduler tagging :* Several fair queueing schemes compute and associate some form of per-packet tag based on which packets are scheduled for service. This corresponds to the finish number in WFQ or the start and finish tags in SFQ. Schemes which use an enhanced form of round robin service (such as WRR or DRR) do not suffer this overhead.

3. *Flow state :* All fair queueing schemes necessarily maintain some form of per flow state which is usually associated with the active queues. A queue/flow is often considered active only if it is backlogged (has packets in the system).

4. *Scheduling for service :* Schemes which associate tags with packets or queues require a mechanism to schedule a packet for service based on the tag value. This may require scanning the per-flow queues to find the queue with the appropriate tag or the maintenance of some form of sorted data

structure.

5. *Buffer management/Drop policy :* Another overhead associated with fair queueing that is often overlooked is the effect of the buffer management policy. For a finite buffer space, a fair queueing policy must also manage or divide buffer space fairly among competing flows. On buffer overflow, the packet discard policy thus becomes important. The implementation overhead associated with a fair drop policy thus comes into play.

We look more closely at each of these overheads and describe efficient algorithms for their implementation and how each of these factors scale.

At first glance, it appears that the *flow classification* overhead is similar to the routing overhead suffered by gateways. We begin by assuming that matching of the header fields required for flow classification can be done in constant time. This overhead is suffered for each active flow in the system, since in a brute force method we match the header of an arriving packet with each active flow in the system. It is easy to do better than this, by using a hash function based on the fields of the header to index into the classification table. Commonly used hashing functions may be found in [20] [16]. In the case of collision some standard techniques such as chaining [19] may be deployed. Clearly, in this case the performance is closely tied into the definition of the hashing function and the traffic patterns. We denote the per-packet overhead in flow classification by F. For hashing combined with chaining the worst case performance is defined by :

$$F = a * n_i$$

where a is the constant coefficient of the classification algorithm and n_i denotes the number of flows that map to the same hash bucket and is a non-deterministic function of the number of active flows and the hashing function. If we use a binary tree search on the hash bucket or sub-table, then we may improve this to $O(log_2(2 * n_i))$. However, the index on which such a binary tree would be based on is not apparent. For varying flow definitions we may be unable to do better than in the linear case.

We note that a scheme such as Stochastic FQ maps colliding flows into the same queue and hence has a constant per-packet overhead of a. In general, the coefficient n_i may be improved by the hash function, a larger number of hash buckets and by higher levels of flow aggregation (and hence fewer active flows).

Scheduler tagging is computationally complex in schemes such as WFQ, but succeeding schemes such as SFQ tag packets in O(1) time. Again, this is a per packet overhead.

The *Flow state* scales linearly with N, the number of active flows in the system. The constant coefficient denoting the amount of state maintained per flow depends on the specific algorithm. Flow state is a memory overhead rather than a computational overhead and becomes important in hardware implementations such as those described in [12]. Flow aggregation helps minimize the flow state by reducing the number of flows.

The overhead involved in scheduling a packet for service is similar in schemes that use some form of tagging for packet reordering. In [18], the author suggests the use of per flow queueing with a double heap for implementing the packet scheduling and buffer management mechanisms. In general, the cost of scheduling a packet in such an implementation has a worst case performance of $O(log_2 2N)$. This derives directly from the heap property, where a heap insertion costs $O(log_2 N)$ and removing the infimum of heap values costs $O(1)$.

The *buffer management* strategy varies across algorithms. Several fair queueing schemes approximate the strategy proposed in WFQ, where packets are discarded in the order of maximum finish number, to make room for the arriving packet. Such a computation requires another heap sorted on the maximum finish time and since finish numbers grow monotonically within a flow, this requires an $O(log_2 2N)$ overhead. In Stochastic FQ and DRR, the buffer stealing approach is deployed, where a packet is discarded from the longest queue. To implement buffer stealing, the author in [16] deploys an array of doubly linked lists, one element for each integral number of elements. All the computations in this implementation are O(1), but the memory requirements make a hardware implementation difficult. Also for each arriving and departing packet, two doubly linked and one singly linked list operations is required. In the case of buffer overflow an additional two doubly linked and one singly linked list operations are required. Such an implementation does not account for varying packet lengths across different flows.

Scheduling Discipline	Flow Classification	Tag Computation	Flow State	Scheduler Service	Buffer Mgt
WFQ	$O(log_2 2n_i)$	$O(N)$	$O(N)$	$O(log_2 2N)$	$O(log_2 2N)$
Stochastic FQ	$O(1)$	NA	$O(N)$	NA	$O(1)$
DRR	$O(log_2 2n_i)$	NA	$O(N)$	NA	$O(1)$
SFQ	$O(log_2 2n_i)$	$O(1)$	$O(N)$	$O(log_2 2N)$	$O(log_2 2N)$

Table 2.1: Comparison of Scheduling Discipline Overhead : The buffer management overhead is not incurred on a per packet basis, but rather scales with the number of discarded packets. The Flow classification overhead listed assumes the possibility of a binary search on a hash bucket in case of collision.

In summary, we furnish Table 2.1 to quantify the overheads of some common fair queueing disciplines.

2.5 Simulation Study of Fair Queueing

This section is organized as follows. We first present an architectural overview of fair queueing in an intermediate node for a deployment perspective. We then describe our trace simulator which builds upon this architecture and attempts to simulate it. The final subsection describes the experiments performed and the results of our simulation study.

2.5.1 Fair Queueing : an architectural overview

Before we attempt to evaluate the feasibility of a fair queueing gateway for deployment in the carrier/backbone network, a clearer understanding of where scheduling fits in the layered architecture is required.

Proposed implementations for fair scheduling and references in the literature

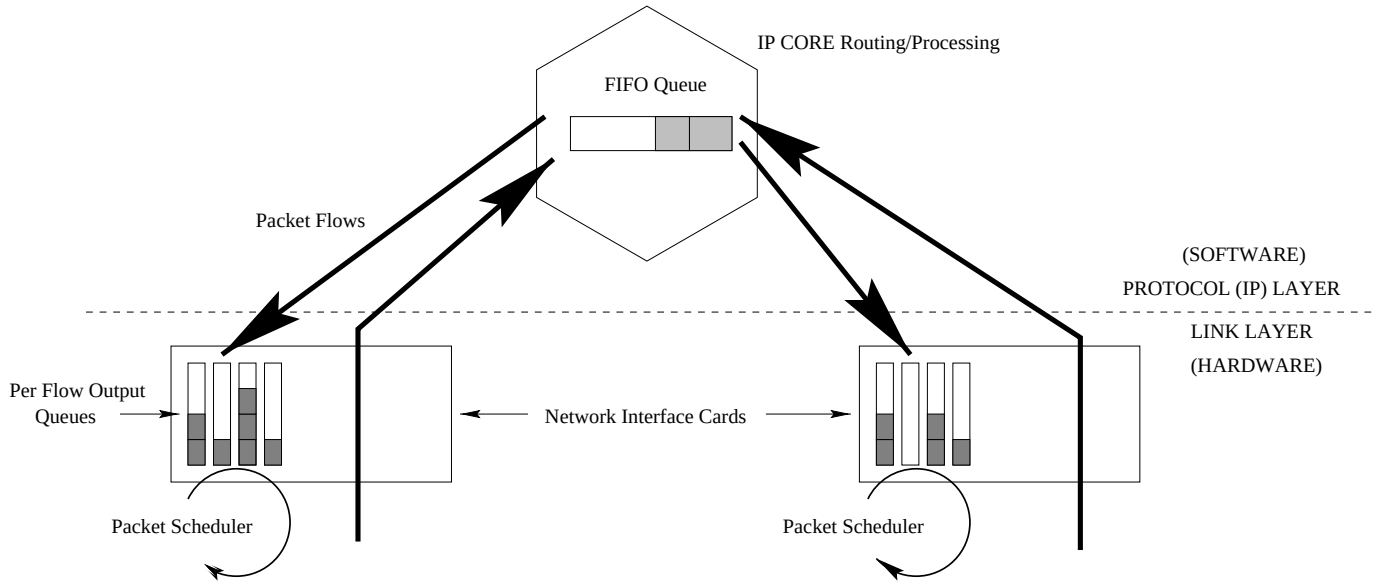

Figure 2.1: Architecture of a scheduling router with two ports. The solid lines represent the path of packets as they traverse the system. The dashed line indicates layering and typical hardware/software boundaries.

[12] [15] generally focus on ATM technology, since applications which require QoS support have been the prime goal of fair queueing algorithms. We attempt a more general description of the deployment of fair queueing.

The components of a generic router include input ports, output ports, a switching fabric and processing element for routing [14]. In general, routers are classified as input queued or output queued based on the comparative bandwidth of the switching fabric and the input ports. Routers with switching fabrics faster than the cumulative bandwidth of the input ports may be classified as output queued routers. In output queued routers, there is no queueing on the input ports, and hence the routing decision must also be made faster than the cumulative bandwidth of the input ports.

We do not look into the problem of accelerating route lookups, or algorithms

for the same. A lot of work has been done in this area in the past and the interested reader is referred to [14] for a survey. Instead we focus on defining and evaluating the cost and algorithms involved in scheduling.

Focusing specifically on Internet Routers, we note that from a layering perspective, the routing decision is taken at the IP layer. Most routers deployed today make this decision in software (or general purpose processors). Cost efficient hardware techniques, specially ones that scale to the requirements of backbone routers, are still an open problem. Scheduling of packets from multiple queues usually assumes an output link/port bottleneck. Hence, packet scheduling is done on a per output port basis. From a layering perspective, per flow queueing may be implemented at the link layer(per output port). Such implementations are traditionally in hardware and are often bundled with the network interface card. An architecture for a scheduling router is depicted in Figure 2.1.

We attempt to study the scalability and feasibility of fair scheduling from the perspective of this architecture. To this end, we have built a trace driven simulator to analyze the component overheads described previously in relation to an implementation scenario such as the above.

2.5.2 The ASQG trace simulator

In this section we describe the salient features of our trace simulator, A Simple Queueing Gateway (ASQG). The simulator attempts to emulate a fair queueing

gateway. An abstraction of the operation of the trace simulator is depicted in Figure 2.2.

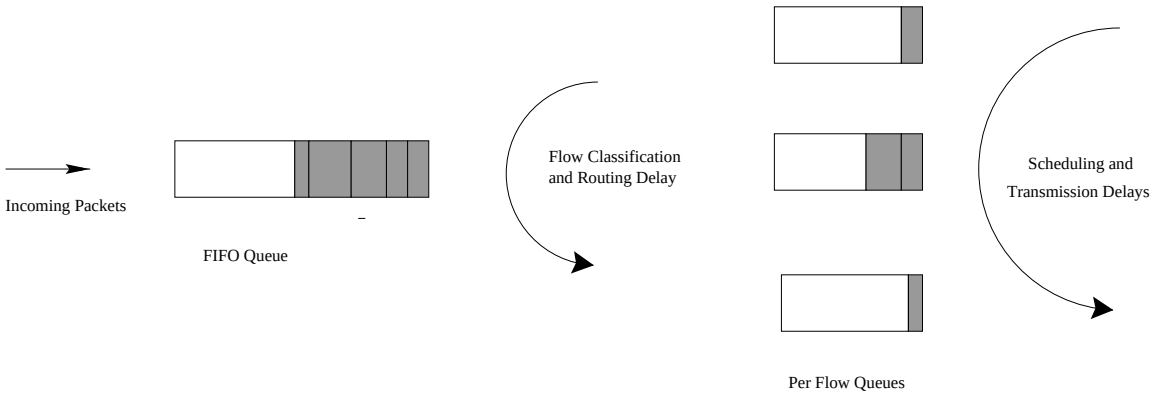

Figure 2.2: An abstraction of the ASQG trace simulator

ASQG is a discrete event trace simulator. It models a many input-single output port forwarding on a router or gateway node. It may be used to emulate both input queued as well as output queued operation. As shown in Figure 2.2, incoming packets are by default placed in a FIFO input queue. The following operations may be characterized on a per packet basis on packets stored in the *input* queue.

1. Routing latency : The latency involved in making a routing decision is modeled as a constant per-packet delay. The inherent assumption is that the size of the routing table is stable.

2. Flow Classification latency: The latency involved in flow classification may be modeled as a constant or as a variable latency. In the case of a variable latency, the latency is computed based on the deployed algorithm. For instance, if Stochastic Fair Queueing is simulated then a constant delay

model is used. For other fair queueing algorithms, the classification delay is represented as $F = a * (log_2(2 * n_i))$, where a is the per packet delay and n_i is the number of entries in a hash bucket. In practice, the n_i is a computed constant which reflects the "goodness" of the employed hashing scheme. The reason for this is twofold. First, our intention is not investigate hashing schemes across different flow definitions. Second, many of the traces we employ in the study are some form of "sanitized" traces (See [23] for the sanitize scripts). For reasons of security and privacy, these traces have been encoded and do not reflect true Internet addresses. Thus hashing comparisons on these traces are unlikely to reflect "real" performance.

3. Switching fabric latency : A constant latency is used to model the memory access and data transfer time required to transfer a packet from the FIFO queue to the output queues.

Note that to emulate input queued routers with ASQG, the sum of the above must be greater than the bandwidth of the input port (i.e. less than the packet interarrival times in a trace).

On completion of flow classification and routing, a packet is "switched" to the appropriate output queue on the output port. Several different flow definitions are supported in ASQG. They include the (source address, destination address, protocol, source port, destination port) quintet, (source address, destination address) tuple, (source, destination network number) tuple (based on the Class

A/B/C addresses), as well as source address, destination address, source network number and destination network number. However, for reasons of sanitized traces, most of our flow classification studies are restricted to work without the network number levels of aggregation. The flow definition mechanism may be passed as a command line parameter.

Once in the output queue, a packet is scheduled for service according to a fair queueing discipline. The following operations are characterized by a corresponding latency on the *output* queues.

1. Scheduling latency : This is a variable latency that is calculated based on the number of currently active flows (scales as $O(log_2 2N)$).

2. Transmission latency : A constant delay is associated in ASQG with the transmission of a packet. This delay is based on the estimate of the transmission line speed that this gateway serves.

All constant delays may be specified at the command line when invoking the ASQG trace simulator. An event queue is used to track discrete event scheduling and progress of time in the simulator.

ASQG also supports specifiable flow timeouts which impact the number of flows and the definition of active flows in a scheduling system. The simulator is intended to be flexible and lends itself to a variety of different studies ranging from flow classification to determining a stable operating point for gateway processing rates.

ASQG is a passive processing element, since the operations of the gateway on the traffic traces do not affect end-to-end performance or modify the end-to- end behavior. Hence, ASQG is not appropriate for the evaluation of end-to-end performance metrics such as the throughput or end-to-end delay. However, it proves to be a suitable tool for studying the performance of a gateway under a given traffic load and to analyze the feasibility of deployment of a gateway with a given set of specifications.

2.6 Some Trace Analysis and Results

We study the following aspects with the help of the trace simulator. Our prime focus is on recommendations for deployment based on a feasibility analysis. We first determine the effects of flow definition and flow timeout as a means of reducing state and processing time in a fair queueing router. As an aside, we motivate the need for scheduling with some observations on the trace data. The implication of the equations embodied by table 2.1 is then discussed in the light of preceding results. We conclude with some final notes on the scalability and deployment of fair queueing gateways.

The traffic traces used in this section are derived from the Internet Traffic Archive(ITA) [24] and from NLANR [25]. These traces are available for free download.

2.6.1 Flow timeouts

The view of a flow at an intermediate router or gateway node is open to interpretation. Specifically in the context of QoS based and hybrid IP/ATM systems, a flow is often specified with an associated flow timeout. A flow is considered to be active for *flow timeout* seconds even if it has no packets queued at the intermediate node. If a packet from this flow arrives in this interval, the flow timer is reset.

Some fair queueing schemes require maintenance of state information even when (temporarily) the flow has no packets in the buffer. Networks which reserve resources based on some form of signaling protocol also maintain state for temporarily inactive flows. In such systems, flow state is maintained until an explicit connection teardown is received or some form of flow timeouts are used.

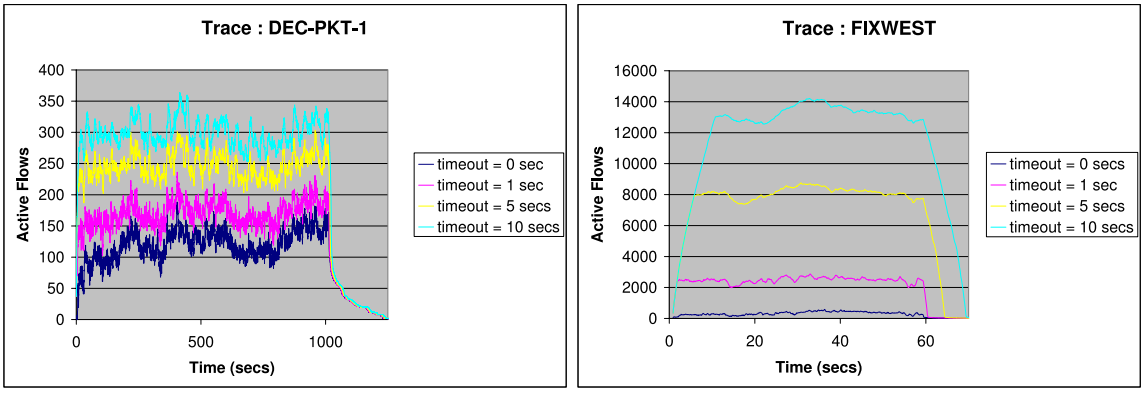

Figure 2.3: Effects of variation in flow timeout values

An obvious way to minimize flow state overhead as well as the number of active flows is to reduce the timeout constraint. An extreme timeout of 0 may be deployed by fair queueing gateways in order to minimize overhead. However, it is

of interest to note that the reduction in number of flows comes with the caveat that the number of flow *operations* increases proportionally. We define a flow operation as a flow deletion/creation. Figure 2.3 depicts the variation in the instantaneous number of active flows for differing flow timeouts. A sampling interval between 100 and 200 ms. is used to collect all data in this chapter.

Trace : Corporate gateway(dec-pkt-1.tcp),Sample size : 500000 packets

Timeout(secs)	Max Flows	Flow Operations	Collisions	Hash fn. efficiency
10	364	9247*2	6590	0.986820
5	304	13417*2	4376	0.991248
1	238	32741*2	3237	0.993526
0	192	83904*2	2738	0.994524

Trace : Backbone router(FIXWEST),Sample size : 500000 packets

Timeout(secs)	Max Flows	Flow Operations	Collisions	Hash fn. efficiency
10	14220	57892*2	229222	0.541502
5	8792	70881*2	220902	0.558144
1	2913	105054*2	177489	0.644980
0	589	178481*2	113173	0.773627

Table 2.2: Effects of inactivity timer variation : While the max flows decreases, the number of flow operations scale proportionally. The flow definition used is flow type 1.

Table 2.2 depicts the increase in flow operations with the decrease in flow timeout values. We look closer into the overhead incurred in a flow creation/deletion versus that of a flow classification. We note that both operations incur a similar search cost, as previously described. In the case of a flow operation, an additional $O(1)$ cost is incurred in creating or deleting state. The large number of such flow operations required suggests that this may still be a significant factor. However, in view of the benefits accorded by a reduction in the number of flows, this overhead is justified. Another point of interest in this

regard, is the overhead incurred with the maintenance of per flow timers for the case when a flow has a finite timeout. This may be non-trivial with large numbers of flows. A timeout of 0 also avoids this overhead.

The numbers for the FIXWEST backbone router trace indicate that in the presence of finite flow timeouts, the gateway would have to deal with an extremely large number of flows. Such a routing box would be expensive to build. A reduction in the timeout values for flows pays big dividends in terms of the number of flows that a backbone router needs to handle.

In summary, we conclude that for a gateway operating in the absence of a reservation mechanism an inactive timeout is an unnecessary overhead. The cost of timer maintenance in addition to the increase in number of active flows may be avoided. Schemes such as Weighted Round Robin(WRR), which require mean packet length estimates, would necessarily have to re-estimate mean packet lengths in the absence of a non-zero flow timeout.

Another point of interest is the deterioration of efficiency of the hashing function with the active number of flows. This demonstrates that the flow classification overhead is a "strong" function of the hashing function and a "weak" function of the active number of flows [1]. The absolute efficiency numbers may be biased due to the sanitized nature of the utilized traces.

[1] This dependence of the flow classification overhead on the active number of flows is not easily characterized and is not reflected in table 2.1.

Trace : Corporate gateway(dec-pkt-1.tcp),Sample size : 500000 packets

Flow Definition	Max Flows	Flow Operations	Collisions	Hash fn. efficiency
f3	24	194585*2	0	1
f2	42	154524*2	0	1
f1	192	83904*2	2738	0.994524
f0	269	83174*2	74691	0.850618

Trace : Backbone router (FIXWEST),Sample size : 500000 packets

Flow Definition	Max Flows	Flow Operations	Collisions	Hash fn. efficiency
f3	47	187582*2	2818	0.994363
f2	58	179047*2	4582	0.990835
f1	589	178481*2	113173	0.773627
f0	795	191026*2	116982	0.766008

Table 2.3: Effects of flow aggregation : The hash function efficiency is across flow definitions and reflects the goodness of the hashing function for each flow definition.

2.6.2 Flow aggregation

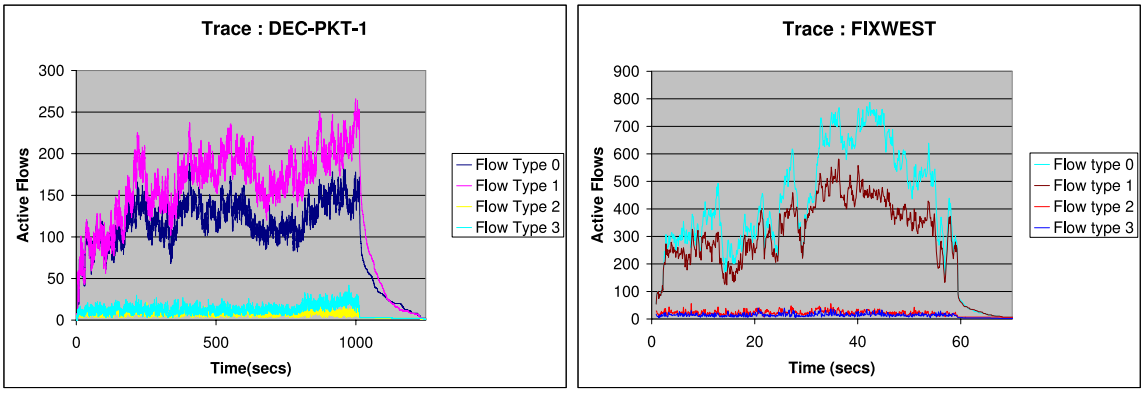

Figure 2.4: Effects of flow aggregation

Another option to minimize flow state is to play with the level of aggregation or granularity of a flow. Such a discussion is closely tied in with the definition of fairness to be applied, as well as to the various prioritized levels of service desired. We note in passing that coarse flow definitions may be used to differentiate between flows in a particular service class, in a hierarchical structure. A definition of such a structure is closely tied in to system

requirements and is beyond the scope of this discussion.

We experiment with several different flow types :

1. Flow Type 0 : Source and Destination Protocol Address, protocol number and source and destination port pair.

2. Flow Type 1 : Source and Destination Protocol Address.

3. Flow Type 2 : Destination Protocol Address.

4. Flow Type 3 : Source Protocol Address.

Several interesting observations may be made from Figure 2.4 and table 2.3. Primarily, we note that the effect of flow aggregation is more marked than that of flow timeouts in the case of the corporate gateway (DEC-PKT-1). Definitions such as source protocol address, while providing some form of fairness, limit the number of flows significantly. Clearly, some isolation may be achieved with such aggregated flow definitions, between misbehaved and well behaved end systems.

Another interesting point is the performance improvement in the hashing function with flow aggregation. Effective hashing schemes can be found which map a single protocol address field to a non-colliding hash bucket. This explains the decrease in number of collisions with flow aggregation in this case. While the absolute numbers are presented with the familiar caveat of sanitized traces, they point in the right direction.

A point of academic interest which we also came across in the course of our flow analysis is that fewer flows result from source protocol address flow definition than in the case of destination protocol address classification. This points towards a few active sources at any given time instant communicating with a larger destination space.

In summary, a flow aggregation level such as a protocol source address is recommended in systems which need minimal service differentiation or need only to ensure fairness between competing sources. It may also be used within a service class to ensure fairness between the flows of that class, in a hierarchical system.

2.6.3 Some more about the traces

Trace Name	Sample size	TCP packets	UDP Packets	Other
FIXWEST	1008918	270643(26.8%)	409160(40.55%)	329115(32.6%)
DEC-PKT-1	3362373	2153462(64%)	829759(24.6%)	379152(11.2%)

Table 2.4: The packets by protocol in some of the traces used

A significant percentage of non-adaptive packets were observed in several of the traces studied. Table 2.4 indicates this trend especially in the backbone router trace FIXWEST. A large percentage of packets in the other category were noted to be General Routing Encapsulation (GRE) packets (about 24% of the total, in fact). The numbers for the FIXWEST router are not representative of the general trends that we observed. However, the statistics for the DEC-PKT-1 trace are

representative and demonstrate a high percentage of non-adaptive UDP packets.

These results provide another strong motivation for the separation of packets into logical flows. Adaptive or well-behaved flows require some method of isolation from non-adaptive flows. Some form of per-flow queueing is therefore essential to ensure fair resource sharing.

2.6.4 Overhead evaluation

We now attempt to compare the implementation cost incurred by some common scheduling disciplines. To this end, we draw up on the analysis of section 2.4.

Each of the overheads in Table 2.1 is assigned an equal weight. Thus, we ignore the constant cost coefficient associated with each of the overheads and focus instead on its scalability. Such a simplification is valid for a relative cost comparison. Also, we note that accounting for constant coefficients in terms of say, number of executable instructions per operation, must take into account specific machine architectures.

The computed cost includes the computational cost (such as the cost of scheduler tagging) as well as the cost of memory (maintenance of flow state). The cost function is computed as follows :

1. Traces of the active flows versus time from ASQG are fed into the cost function. The equations of Table 2.1 are then applied to these traces to compute the Tag computation, flow state and scheduler service overheads.

2. The flow classification overhead is computed using the average efficiency of the hashing function over a simulation run and the average number of colliding flows over that simulation run. We assume a linear search over the colliding flows unlike in Table 2.1.

Quantification of the buffer management overhead requires assumptions about buffer size on the gateway and scales with the number of discarded packets. Hence, this is not accounted for in the cost function.

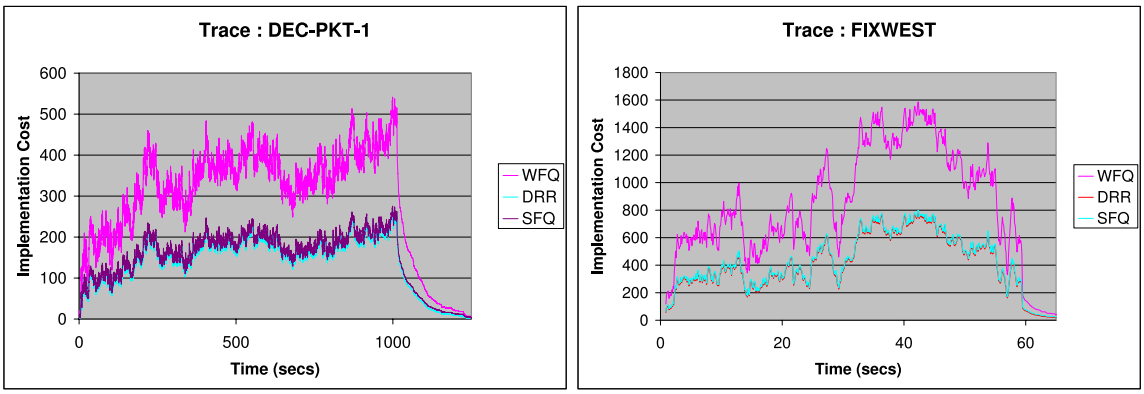

Figure 2.5: A comparative per packet overhead evaluation for some common scheduling disciplines.

Figure 2.5 shows the computed cost for Weighted Fair Queueing(WFQ), Deficit Round Robin(DRR) and Start-time Fair Queueing(SFQ) over the dec-pkt-1 and fixwest traces. The numbers on the Y-axis represent the number of operations required for each arriving packet in the sampling interval. The flow definition used was flow type 0.

The following observations may be made from the traces. WFQ has the highest implementation cost and a scheme such as SFQ which affords comparable

performance should be chosen over it. More interestingly, we note that SFQ and DRR appear to differ only slightly in cost. However, we note that this cost is incurred on a per packet basis and small differences in cost represent large performance drops in terms of processing ability.

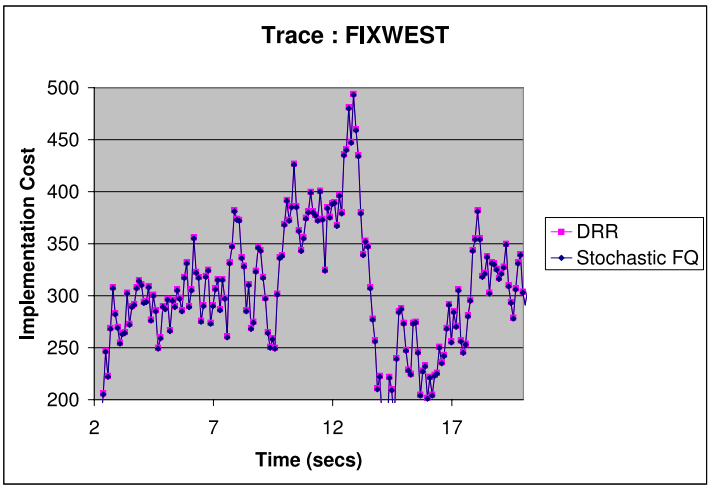

Figure 2.6: Implementation cost of DRR vs. Stochastic FQ. Only a minimal performance gain is achieved by Stochastic FQ over DRR.

It may be noted from 2.6 that even with relatively poor hash function performance (see Table 2.4), very little performance gain is achieved by Stochastic Fair Queueing over DRR. This trend was observed over a large number of traces studied. Hence, DRR may be chosen over Stochastic FQ since it has marginally higher implementation cost for improved performance in terms of fairness and bounded delay.

Further, the cost implementing a scheme such as Deficit Round Robin over that of simple FIFO queueing can be localized to two factors, the flow classification and flow state overheads. From our trace studies, we have shown

that the flow classification overhead, given a good hashing scheme, is not excessive in the light of recent strides in computer processing power. Also, optimizations such as Stochastic FQ eliminate this overhead almost completely. With the reduced cost of memory, maintenance of per flow state is another overhead which should not be an enduring obstacle to the deployment of fair queueing.

2.7 Conclusions and Future Work

In this chapter, we have attempted to present a framework within which the feasibility of deployment of fair queueing for a particular set of system requirements can be evaluated. The various issues that need to be considered for such an evaluation have been analyzed in detail, with the trace analysis presenting concrete examples and quantitative results based on live traffic.

The choice of a fair queueing discipline for a system is usually based on the characteristics required for the system in terms of the delay and fairness bounds of the discipline. In this chapter, we have presented methods and sample analysis to evaluate the implementation feasibility of such a system. A discussion on efficient implementation algorithms to minimize fair queueing overhead is summarized in Table 2.1 for several common fair queueing disciplines. In conjunction with the trace analysis in section 2.6.4, this provides an effective quantification of the expense involved in the deployment of a fair queueing

discipline.

We note from the results presented in the above that schemes such as Weighted Round Robin or Deficit Round Robin are easily implemented with a small increase in overhead over FIFO queueing. Hence, the implementation of such disciplines is feasible even with the performance demands of the current Internet.

Sections 2.6.1 and 2.6.2 outline methods to minimize fair queueing overhead independent of the fair queueing discipline. A reduction in flow timeout values can significantly reduce the maintenance of flow state as well as the number of active flows. The benefits incurred from such a reduction outweigh the corresponding increase in the number of flow operations.

Flow aggregation techniques which minimize flow state are also presented in this context. While flow aggregation is usually at the cost of coarser service differentiation, flow aggregation can be used within service classes in a hierarchical scheduling system to minimize overhead.

Section 2.6.3 provides additional motivation for some form of fair queueing by noting that a sizable percentage of flows in the Internet are non-adaptive and hence the need for fair queueing to provide isolation between well-behaved(adaptive) and misbehaved sources.

Significant problems for future research include : Development of analytical methods for performance analysis of these fair queueing algorithms; Consideration of other performance metrics and a systematic trade-off analysis

and investigation of performance under realistic traffic models (e.g. self-similar and multifractal).

Chapter 3

Buffer Management Strategies for Per Flow Queueing Hybrid Gateways

3.1 Introduction

In Chapter 2 we analyzed the feasibility of deployment of fair queueing techniques on gateways and routers in the Internet. We contend that fair queueing in the absence of effective buffer management strategies is inadequate in optimizing traffic behavior at the intermediate node. While fair queueing ensures fair and efficient use of network bandwidth, the role of buffer management becomes evident during periods of network congestion. Congestion recovery and congestion avoidance in the Internet usually involve packet discard. Packets must be discarded in accordance to a policy that satisfies the following criteria.

1. Isolation and fairness: A well-behaved source must be protected from sources that are mis-behaved or exceed their "fair" share of the buffer.

Fairness refers to the dropping of packets from a flow in proportion to the buffer occupancy of the flow.

2. Throughput : While being fair, a buffer management strategy should allow high link utilization and end-to-end throughput.

3. Implementation Efficiency : The policy should be simple and easy to implement and its running cost/overhead should be low.

In this chapter, we explore various buffer management schemes in the context of both fair queueing and hybrid gateways. Several limitations associated with existing approaches to buffer management are exposed. We then propose Probabilistic Fair Drop (PFD), a new approach to buffer management specifically targeted at our problem space.

Throughout this chapter, our primary focus is on optimizing the behavior of adaptive transport layer protocols, that react to congestion notification. The method of congestion notification assumed is packet discard. We use the Transmission Control Protocol (TCP) [37], the most commonly used protocol for reliable data transfer on the Internet, to illustrate our discussion and in our simulations.

TCP is an adaptive window based protocol that performs flow and congestion control. The rate of a TCP source is controlled by a sliding window whose size varies in response to acknowledgements, timeouts and packet losses. The growth of the sliding window is in response to acknowledgements received from the

remote receiver. Thus, the sending rate of a TCP source is tied closely to the round trip time between the source and the destination of a TCP connection.

When multiple TCP connections share a bottleneck link with a high-bandwidth delay product, it has been demonstrated that smaller RTT connections capture most of the network bandwidth at the expense of long RTT connections since they can step up their sending rates more rapidly [36]. The cumulative nature of the TCP acknowledgements allows the discovery of only one packet loss per window of data. Hence, multiple packet losses result in highly degraded performance and low link utilization.

The inherent unfairness of TCP to connections with long RTTs is compounded when a connection that traverses a long latency satellite hop shares a bottleneck link with other connections that have relatively smaller RTT's. Our specific focus in this chapter is to optimize performance for such a setup as illustrated in Figure 3.1.

In the following section, a critical review of existing approaches to buffer management and an exposition of some of their limitations is offered. The PFD algorithm is then outlined in the next section. We compare the performance of PFD with existing schemes by means of simulation and present the results in subsequent sections. Some further optimizations to improve the performance of long RTT connections are then discussed and simulated. Some efficient implementations for PFD are discussed in the following section. Most of the work presented in this chapter was conceived in conjunction with the authors in [26].

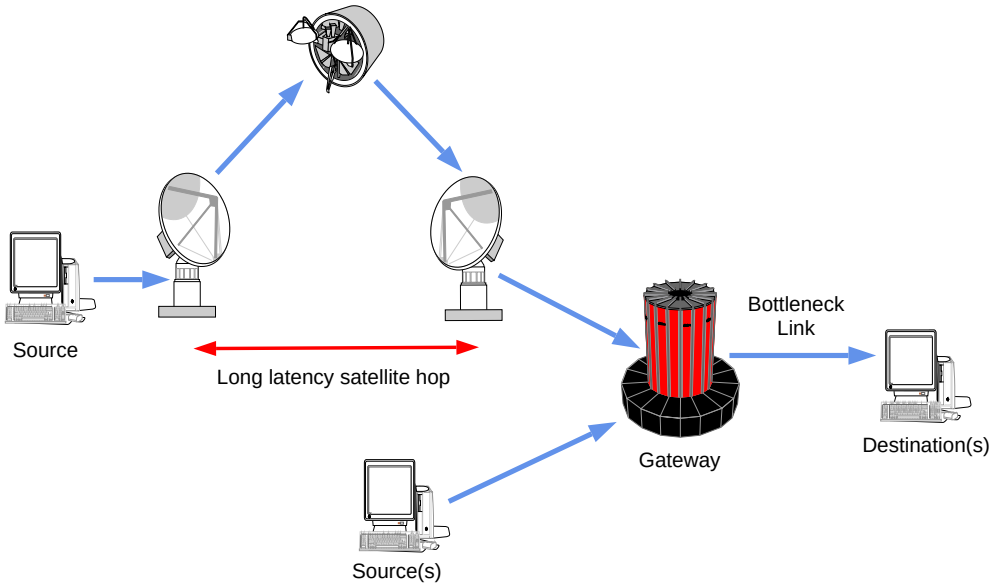

Figure 3.1: Hybrid System of Interest. Both short RTT terrestrial connections and long RTT satellite connections traverse the bottleneck gateway.

3.2 Related Approaches to Buffer Management

Most gateways deployed in the current Internet do not explicitly manage their finite buffers. In the case of congestion, packets that arrive when the buffer is full are simply dropped. This lack of policy is commonly referred to as "drop-tail". While such a stateless scheme offers great ease of implementation and preserves the "best-effort" nature of the Internet, it has several undesirable effects.

The authors in [32] outline the effects of strongly periodic network traffic on deterministic buffer management schemes such as drop-tail and demonstrate the resulting bias towards some connections. [32] also demonstrates that drop-tail is biased towards bursty connections. The inherent bias towards connections with large round trip times is attributed to the window increase algorithm of TCP. The authors note that the throughput increase of a TCP connection is roughly

r^{-2} pkts/sec every second, where r is the round trip time of the connection. They propose a window increase algorithm where a node increases its window by $c * r^2$, which results in a constant increase in throughput of c pkts/sec for every second, for some constant c.

Random drop attempts to alleviate the bias due to traffic phase effects as well as the bias towards bursty connections. In Random drop congestion recovery, a packet is discarded at a random position in the buffer on buffer overflow. Among the drawbacks of this scheme are its extremely expensive implementation costs. A congestion avoidance mechanism was also proposed with Random Drop [27]. This involves a selection of incoming packets to be randomly dropped at a rate that depends on the operating point of the gateway.

Early Random Drop (ERD) [28] attempts to randomize the buffer management policy by randomly dropping a packet from the buffer with a fixed probability when the instantaneous buffer size exceeds a specified threshold. Thus Early Random Drop also attempts to anticipate congestion by dropping packets before buffer overflow. However, the simplistic fixed parameters in Early Random Drop do not work well in all cases and more sophisticated early discard schemes were called for.

Random Early Detection (RED) [29] gateways attempt to detect incipient congestion by monitoring the low pass filtered *average queue size* . The random drop notion of RED built on previous work on Random Drop and Early Random Drop gateways. RED attempts to alleviate traffic phase effects as well as the

unfairness to bursty connections.

The decision to accept an incoming packet in RED is based on whether the average queue size exceeds or conforms to predetermined thresholds. If the average queue size is between min_{th} and max_{th} then the arriving packet is dropped with a probability that is an increasing function of the average queue size. If the average queue size exceeds max_{th} then arriving packets are dropped with probability one. RED attempts to maintain a low average queue size while admitting occasional bursts of packets. The goal of RED is to drop packets at evenly spaced intervals to avoid both an unfair bias towards some connections as well as global synchronization. Some form of per flow fairness is ensured in RED since the packet drops experienced by a connection during congestion is roughly proportional to its buffer occupancy. The randomness of the drop decision combined with the dropping of the incoming packet in RED could cause the congestion window of a flow to be halved even if it does not exceed its fair share. RED implementations may suffer from short term unfairness.

Flow Random Early Detection (FRED) [30] was motivated by the unfair bandwidth sharing of RED gateways between adaptive and non-adaptive flows. FRED imposes per-flow accounting and the loss rate of a flow in FRED is proportional to the flow's buffer use. FRED introduces per flow parameters min_q and max_q, which are the minimum and maximum number of packets each flow is allowed to buffer FRED also defines *avgcq*, a per flow buffer utilization estimate. Flows which exceed *avgcq* are penalized over flows which have less than *avgcq*

packets enqueued. For simplicity, *avgcq* is defined as *avg* by the number of active flows. The general RED algorithm is preserved in FRED, with the low pass filter average estimated both at packet arrival as well as departure.

FRED provides greater flow isolation than global schemes such as RED. However, the dropping of the incoming packet is still preserved in FRED and when a drop decision is evaluated, only the queue pertaining to the arriving packet is examined. A rigid allocation of thresholds and a parameter explosion is associated with a scheme such as FRED.

The Drop from Front strategy proposed by the authors in [35] is based on the interaction of the buffer management policy with the fast retransmit mechanism of TCP [38]. In fast retransmit, a TCP segment is considered lost if three duplicate acknowledgements are received and the segment retransmitted. If a packet is discarded from the head of the buffer, then duplicate acknowledgements are sent one buffer drain time earlier, and the dropped segment retransmitted earlier. Thus, in the presence of TCP's fast retransmit mechanism, the drop from front strategy achieves some performance improvements over the usual drop from tail mechanism.

The attention focussed on Fair Queueing strategies and their efficient implementation motivated the authors in [31] to consider buffer management strategies applicable to a fair queueing scenario. The authors propose the use of "soft" per flow buffer thresholds to dynamically share the total available buffer space among the active connections. In the case of buffer overflow, a packet is

discarded from the connection which exceeds its fair share (defined as B/n, where B is the global buffer space and n is the number of active connections) the most. The drop policy, termed Longest Queue Drop (LQD) discards the packet at the head of the queue, thus leveraging TCP's FRR mechanism [35]. LQD uses the instantaneous queue size to determine non-conformance. A variant of LQD, Random LQD (RND) is also proposed. RND picks a queue at random from the non-conformant queues and discards a packet from the head of that queue.

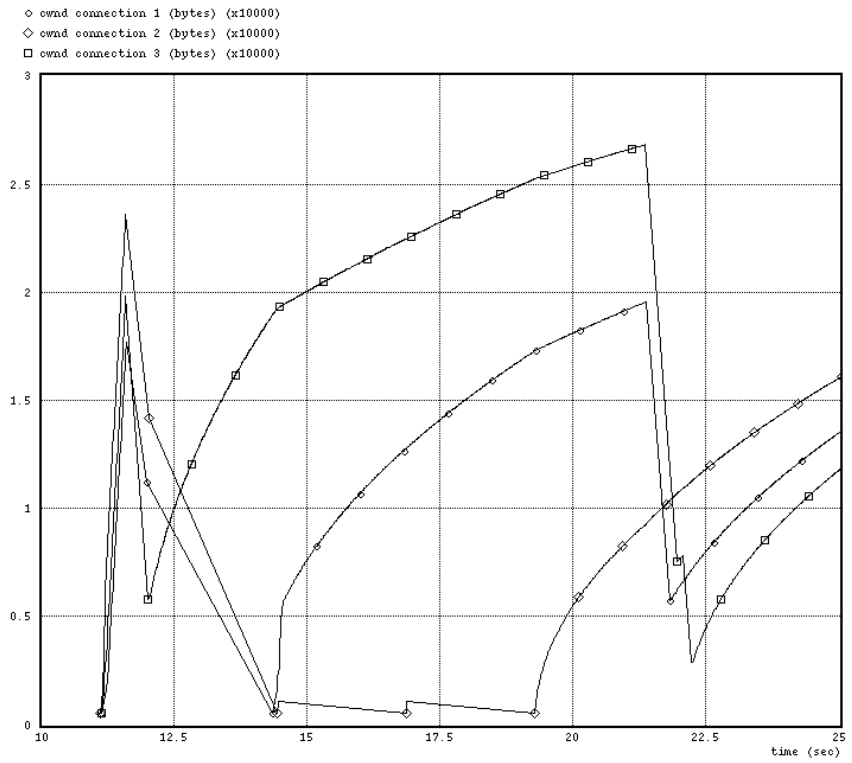

Figure 3.2: Multiple drops in RND : connection 1 and connection 3 suffer drops with high temporal locality and drop their windows simultaneously. connection 2 suffers multiple drops initially and experiences severely degraded performance.

LQD and RND do not attempt to detect incipient congestion and are forced to drop packets when the buffer overflows. It is to be noted that the contents of

the gateway queues when the buffer overflows are not necessarily representative of the average traffic through the buffer. Strongly periodic traffic may bias a drop mechanism such as LQD.

By waiting until the onset of congestion, it is highly likely that multiple packet drops occur with high temporal locality, leading to one of two undesirable behaviors.

1. The same source is penalized multiple times, leading to severe degradation in the throughput of this source (LQD).

2. Multiple sources are penalized, causing flow synchronization and hence degradation in the link utilization(RND).

Such behavior is more likely to occur in flows which are slow to react, such as long RTT flows. In summary, and to motivate our approach, we make the following observations on existing buffer management schemes.

1. Predicting the onset of congestion helps avoid multiple drops with high temporal locality.

2. Per flow schemes such as FRED however, are rigid in their threshold allocations, and do not allow a flow to expand into the global buffer space even when buffer space is available.

3. Schemes with a global threshold such as RED are unfair to low rate TCP flows and in the short-term.

3.3 The Probabilistic Fair Drop (PFD) Algorithm

In the presence of per flow queueing, we dynamically allocate the total available buffer B equally among all currently active connections n. A soft threshold b_i is allocated for each flow $b_i = B/n$. Later sections discuss buffer dimensioning to alleviate TCP's unfairness to long RTT connections.

We attempt to predict the onset of congestion by monitoring the total instantaneous buffer occupancy $q\_size = (\sum_{i=1}^{n} q_i)$ against a single global threshold, $thresh$. As long as the buffer occupancy does not exceed $thresh$, no packets are discarded and a flow may expand to fill all the available buffer space. Once $thresh$ is exceeded, a drop decision is executed with a fixed probability p. This decision is independent of the choice of the flow to be penalized.

If the decision is made to discard a packet, a flow is chosen as follows. We define a normalized instantaneous flow size, $n_i = q_i/b_i$. We now choose the flow with the highest normalized instantaneous flow size and *push-out* the packet at the head of the chosen flow.

The drop probability p and threshold $thresh$ prove to be effective measures to counter the effects of multiple packet losses for a single connection as well as flow synchronization.

Probabilistic discard with a conservative threshold and a low value of p ensure that a bursty connection will suffer packet losses that are further apart on the time axis, and not be penalized repeatedly. This also allows more time for

the connection to respond to congestion. Unlike RND, where packets necessarily had to be discarded from a set of flows when the buffer overflowed, with PFD, packet discards from different sources occur with low temporal locality, thus preventing flow synchronization.

Specified Parameters :
- p : fixed drop probability
- $thresh$: fixed threshold ($0 < thresh < 1$)
- B : global buffer space

Variables :
- q_size : global instantaneous buffer occupancy
- q_i : instantaneous occupancy of queue i
- n_i : normalized instantaneous occupancy of queue i
- b_i : dynamic soft threshold of queue i
- $drop\_q$: queue to discard from i

Algorithm :

For each packet arrival
- increment q_size
- classify packet into flow i
- increment q_i
- compute $n_i = \frac{q_i}{b_i}$

if $q\_size > B$
- $drop\_from\_longest\_normalized\_q$

else if $q\_size > thresh * B$
- with prob p
- $drop\_from\_longest\_normalized\_q$

else continue

$drop\_from\_longest\_normalized\_q$
- choose $drop\_q = \overset{argmax}{i} \ (n_i)$
- *push-out* the packet at the head of $drop\_q$
- decrement q_size

Figure 3.3: The Probabilistic Fair Drop Algorithm

Packet drops in PFD are not always from non-conformant sources, thus providing an early warning mechanism for conformant sources with large queue buildups as well. Since PFD penalizes the queue which utilizes its largest

normalized buffer share, the drop probability for a queue is proportional to its normalized buffer occupancy.

The choice of the values of parameters $thresh$ and p now becomes an important design consideration. We studied the link utilization resulting from the scheme over a range of values for $thresh$ and p to arrive at the best combination. For all the simulations in this chapter, we adopted a threshold value of 0.2 (20% of the buffer) and a drop probability of 0.02.

3.4 Simulation Methodology & Results

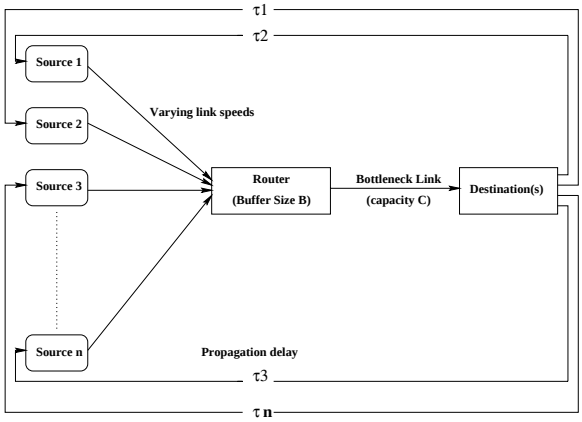

Figure 3.4: Simulation Model

Simulations were carried out using the OPNET Network simulator [41]. We adopt the n source TCP configuration for the network model as shown in Figure 3.4. In this configuration, the TCP sources share a common bottleneck link of capacity C and are subjected to varying propagation delays. The TCP sources implement TCP Reno with the Fast Retransmit and Fast Recovery algorithms [38].

The router at the bottleneck link implements packetized versions of several buffer management strategies including Tail Drop, Drop from Front, RED, LQD, RND and PFD. Scheduling strategies such as FCFS, Round Robin and Start-time Fair Queueing (SFQ) [9] are also configurable at this router. Queueing effects are limited to the IP queues in the gateway by matching the IP forwarding rate at the router with the desired bottleneck link rate C. The Fair Queueing scheduler used implements the SFQ algorithm. The algorithm is scalable and lends itself well to high-speed implementations.

In our simulations, we studied the performance of the PFD buffer management scheme with a FQ scheduler when multiple TCP connections with different rates and propagation delays share a bottleneck link. The router at this bottleneck link is configured with various buffer management and scheduling policy combinations used for the study. We compare the performance of FQ-PFD with FQ-RED, FQ-LQD and FQ-RND.

3.4.1 Performance measures

We evaluate the performance of the various schemes using the TCP "goodput". We define the goodput as the average rate of data delivered to the application by TCP. To evaluate the steady state performance infinite sources are run for "long" periods of time until the average rate stabilizes. In the rest of our discussion, we use the terms throughput and goodput interchangeably.

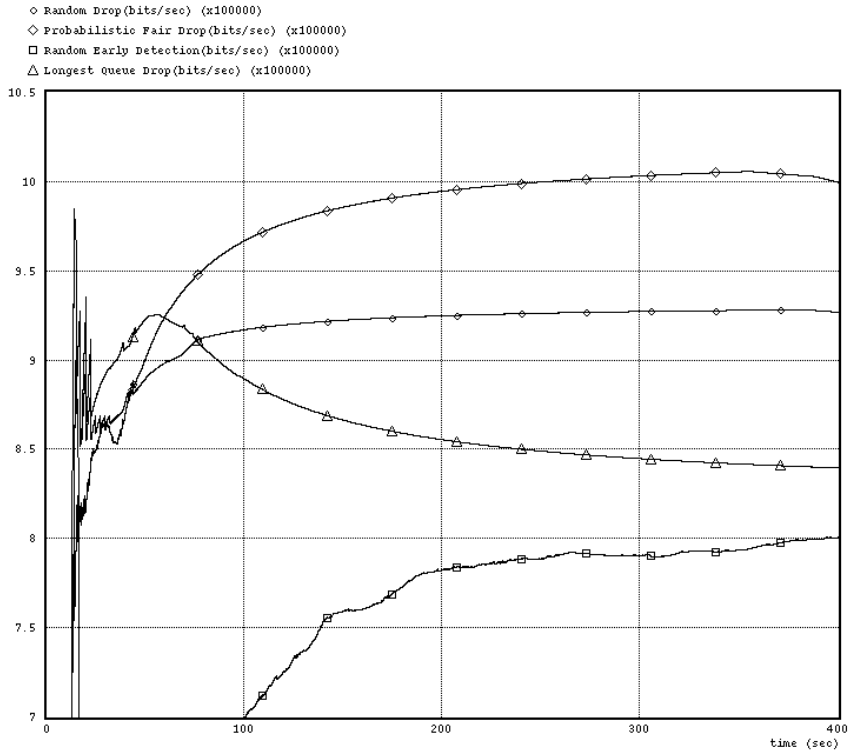

Figure 3.5: Evolution of TCP goodput with time for 10 low RTT TCP-reno connections sharing a bottleneck link of 2 Mbps

The ideal combination of scheduling discipline and buffer management policy would ensure that each flow threading the bottleneck router receives the same amount of service over any interval of time. A good scheme would penalize a flow that exceeds its fair share and ensure that excess bandwidth is shared fairly among backlogged flows when the network is not congested. We use the *Fairness Coefficient* defined in [40] as a measure of how fairly the scheme distributes bandwidth among competing flows. The Fairness Coefficient, F is defined as

$$F = \frac{(\sum_{k=1}^{N} b_i)^2}{n \sum_{k=1}^{N} b_i^2} \tag{3.1}$$

where n is the number of flows and b_i is the bandwidth obtained by $flow\ i$. From

this definition, we see that an ideally fair scheme would have a fairness coefficient of 1, while a completely unfair scheme would result in a coefficient of $1/n$.

3.4.2 Comparison with other schemes

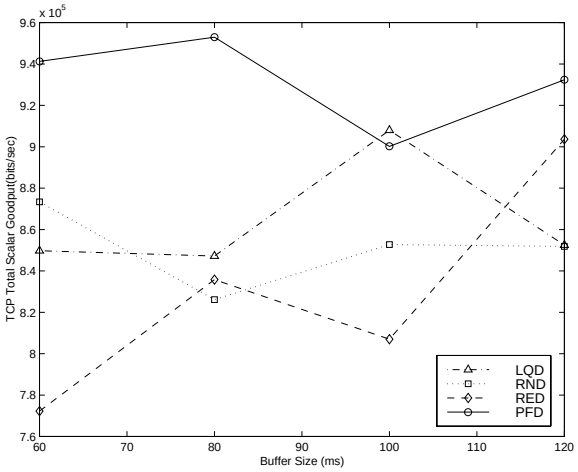

Figure 3.6: TCP goodput (after 500 sec) versus buffer size at the bottleneck router for 10 low RTT TCP-reno connections sharing a bottleneck link of 2 Mbps

We compare the performance of RED, LQD and RND with PFD with an SFQ scheduler. For all the simulations in this section, the TCP sources are modeled as large file transfer applications. Thus, we contrast the steady state performance of PFD with that of the other schemes. From Figure 3.5 we see that in steady state PFD outperforms RED,LQD and RND. The *early warning* system of PFD combined with the pushout drop policy allows PFD to ramp up to a higher average rate.

Figure 3.6 shows that PFD in general outperforms other schemes even in the case of short RTT connections sharing a bottleneck link. The buffer size is

represented as a link-speed equivalent buffer in ms. We note that though the PFD algorithm is targeted for deployment on hybrid gateways, excellent performance is achieved in this setup for connections with propagation delays in the region of a typical WAN round trip time (20 ms). For this simulation setup it was observed that all the schemes have a fairness coefficient close to 1. This may be attributed to the similar propagation delays and source rates of the connections sharing the bottleneck link. In such a scenario, a fair queueing system is adequate to provide fair service even in the absence of a fair buffer management policy.

We now turn our attention to connections with widely varying RTTs. We consider the goodput of 10 TCP-Reno connections with RTTs varying from 20 ms to 200 ms (of the order of a satellite RTT) sharing the same bottleneck link. Note that our version of TCP-Reno also implements the TCP window scale option [39] and hence the goodput of the large RTT connections are not limited by the TCP congestion window growth. We plot the performance over a widely varying range of buffer sizes and attempt thereby to capture the goodput and the fairness metrics over this range. The range of buffer sizes is chosen to capture regions in which the queueing delay is insignificant in comparison to the longest RTT as well as regions in which the queueing delay is significant and is comparable to the RTT of the longest RTT connection.

From Figure 3.7 we see that for lower buffer sizes RND performs better than PFD in terms of goodput. A comparison with the fairness coefficient curve

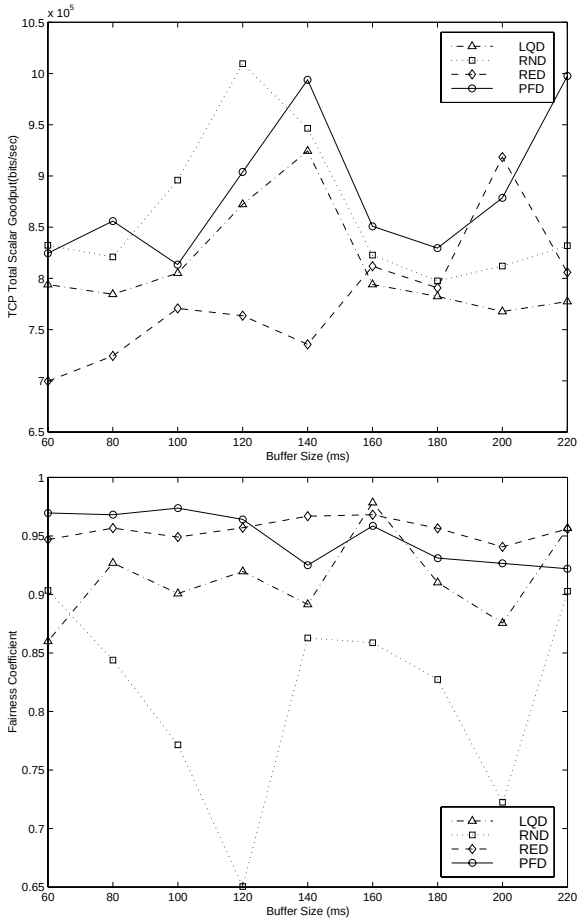

Figure 3.7: TCP goodput and Fairness Coefficient (after 500 sec) versus buffer size at the bottleneck router for 10 TCP-Reno connections with RTTs varying between 20 and 200 ms

indicates, however, that the improved goodput is at the expense of fairness to long RTT connections. This may be attributed to the random choice that RND makes from the set of non-conformant connections when choosing a flow to penalize.

Not surprisingly, LQD is observed to be consistently fairer than RND. PFD's higher fairness coefficient can be explained by observing that while both LQD and PFD are similar in flavor when picking a flow to penalize, the early drop

nature of PFD allows longer RTT flows more time to react, thus guarding against the possibility of multiple drops.

As expected, RED experiences the worst performance in terms of goodput. The high degree of fairness of RED in this case may be explained by observing that like PFD, RED also employs an early detection mechanism and the random choice of flow appears to even out over longer periods of time. Also we note that in the regions of lower buffer sizes, PFD outperforms RED in terms of fairness. With larger buffer sizes, RED's averaging mechanism allows it to maintain low average queue occupancy resulting in fewer drop decisions and hence an improved fairness coefficient. It may also be noted that schemes such as PFD are strictly fair over shorter time scales. Also, for asymmetric channels studied in [31], RED performs very poorly in terms of fairness due to its drop tail nature.

We note that PFD offers the best performance in terms of a combined fairness and goodput perspective. Excellent fairness is achieved in PFD without experiencing a hit in terms of goodput.

3.4.3 Buffer dimensioning

In previous sections we explored buffer management schemes that were tuned to achieve fair link sharing between terrestrial and satellite connections. No a priori knowledge about the characteristics of the connections were assumed. In this section, we redefine the buffer allocation based on the connection's RTT to

achieve a higher degree of fairness. Such information is available to TCP-aware gateways, such as the connection spoofing/splitting gateways, which are the subject of the next chapter.

An analysis of TCP throughput as a function of loss probability [42] has shown that TCP throughput varies inversely as the Round Trip Time (RTT). The dynamics of the algorithm also causes the rate of window growth to be inversely proportional to RTT, both in the Slow Start and Congestion Avoidance phases of the algorithm. This inherent bias towards long RTT connections is further worsened by global buffer management strategies, that do not allow long RTT connections to build up the larger windows they need in order to maintain the same throughput as shorter RTT connections.

We examine the throughput of long RTT connections when FQ is used in conjunction with proportional buffer allocation and PFD. The "soft" per-flow buffer thresholds are allocated in proportion to RTT, with b_i now being allocated as

$$b_i = \{RTT_i/(\sum_{i=1}^{n} RTT_i)\}B$$

This proportional allocation of soft thresholds allows longer RTT connections a better chance to build up their larger windows. PFD is used for packet discard decisions. The assignment of equal weights to all connections ensures that the scheduler still treats all connections with equal priority, giving each a fair share of the link bandwidth.

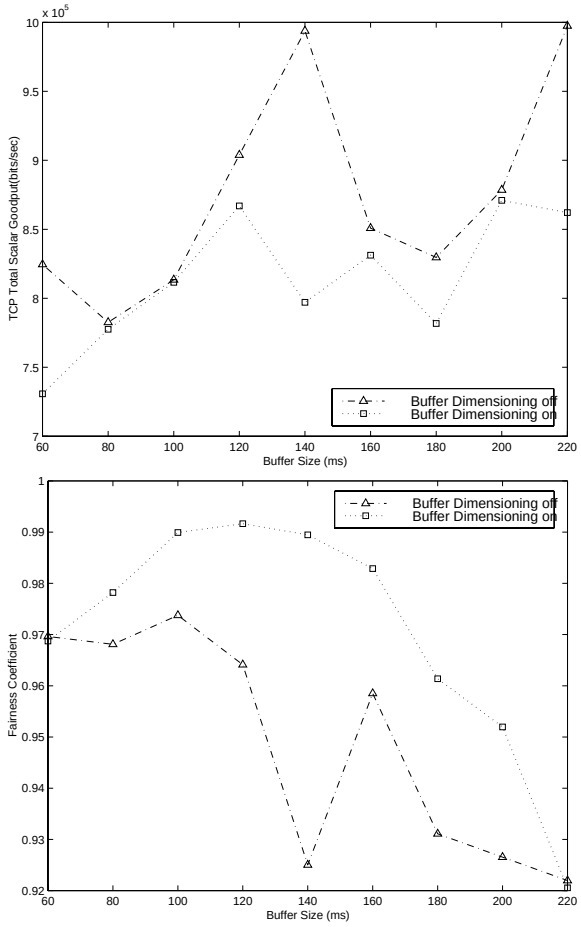

Figure 3.8: TCP goodput and Fairness Coefficient (after 500 sec) versus buffer size at the bottleneck router with and without buffer dimensioning.

The graphs of Figure 3.8 depict these trends. While a decrease in TCP goodput is observed with Buffer Dimensioning, a corresponding increase in fairness towards connections with higher RTTs results in a higher fairness coefficient. In the case of a hybrid gateway, bandwidth is usually purchased from a satellite provider. The pricing is tied into the time for which the uplink bandwidth is purchased rather than on the utilization. In such a scenario, it is economically important for a hybrid gateway to keep the satellite pipe full. In

this context, an overall decrease in throughput, such as the one suffered due to buffer dimensioning, is offset by the increased throughput of the higher RTT connections.

A point of note is the decline of the fairness coefficient with increase in the buffer size. With an increase in the buffer size, there is less pressure on the buffer and hence, fewer discarded packets. In the limit, as the buffer approaches infinity, there will be no discarded packets. In such a case, the only limitation on the throughput of a TCP connection is the RTT of a connection. Thus, even in the presence of perfect fair queueing, shorter RTT connections will have increased throughput over longer RTT connections. This bias is related to the nature of the TCP algorithm and can only be corrected by modification of end system TCP implementations. Such modifications are beyond the scope of this discussion.

3.5 Implementation Considerations

In this section, we discuss the computational complexity of implementing a scheme such as PFD and discuss algorithms for its efficient implementation. We also propose Quasi-pushout PFD, an efficient version of PFD which uses the concept of quasi-pushout cell discarding [43].

Once a decision has been made to discard a packet in PFD, the algorithm picks a queue to penalize based on the normalized buffer occupancy of the queue. The penalized queue is one which has the highest normalized buffer occupancy.

Selection of a queue to discard a packet from is computationally expensive as it may involve a large number of comparisons between all the active queues in the system. The trace analysis of chapter 2 shows that this number may be as large as 795 active flows in a backbone router (FIXWEST).

Among the known efficient implementations of a drop from longest queue scheme is the buffer stealing approach presented in [16]. However, such a scheme is not applicable in the case of PFD due to the non-integer nature of the parameter in consideration, i.e. the normalized buffer occupancy. We borrow the concept of quasi-pushout from the authors in [43]. We term the resulting variant of PFD as Quasi-pushout PFD.

In Quasi-pushout PFD, two additional variables, *discard_q* and *discard_q_occupancy* are maintained. On every enqueue, dequeue and queue discard operation, the normalized occupancy of the active queue is compared to the *discard_q_occupancy*. If the active queue has a higher normalized occupancy, then *discard_q* is set to the active queue and the *discard_q_occupancy* variable is updated with the new value. If the active queue is the same as the *discard_q* then the *discard_q_occupancy* estimate is updated to reflect the new occupancy. Once a decision is taken to discard a packet, the *discard_q* is chosen as the queue to penalize.

Thus, Quasi-pushout PFD trades efficient implementation for the strict fairness of PFD. We evaluate the performance of Quasi-pushout PFD in relation to PFD by simulation. The results are presented in Figure 3.9. As expected,

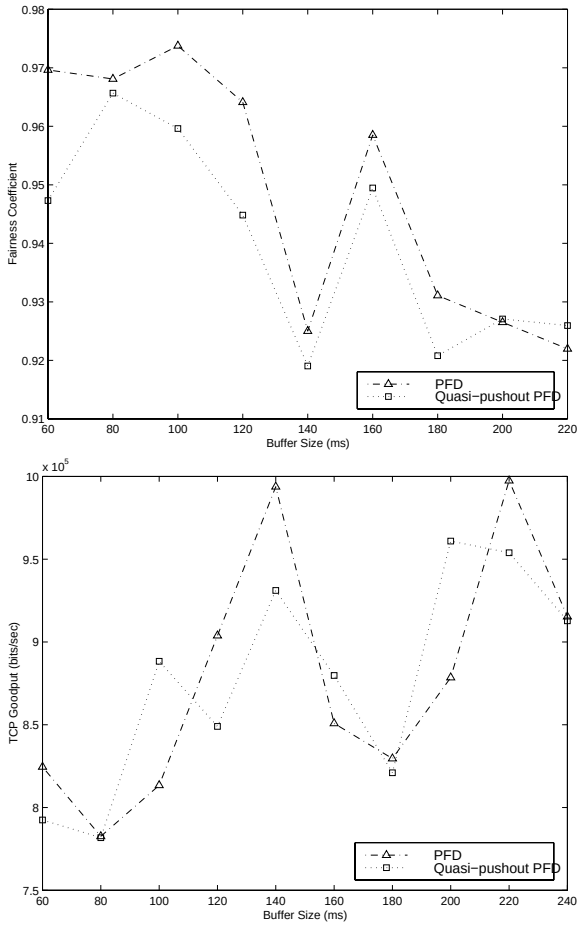

Figure 3.9: TCP goodput and Fairness coefficient of PFD versus Quasi-pushout PFD for 10 connections with round trip times between 20 and 200 ms.

Quasi-pushout PFD does not perform as well as PFD in terms of fairness but still yields a fairness coefficient of at least 0.92 in the simulation setup studied. In terms of TCP goodput, both schemes have comparable performance and no general trends may be observed. Our results indicate that for the sake of implementation efficiency, Quasi-pushout PFD may be chosen over PFD in backbone routers.

3.6 Conclusions and Future Work

In the course of this chapter, we investigated several commonly used buffer management schemes. Some of their limitations prompted the proposal of Probabilistic Fair Drop (PFD), a new buffer management scheme for use in hybrid gateways in conjunction with per flow queueing. The performance of PFD was then compared with other buffer management schemes by extensive simulation. Our simulation results indicate that PFD is best suited for deployment on hybrid gateways, as it has the best performance in terms of a combined fairness and goodput perspective.

Buffer dimensioning based on the individual round-trip times of each connection sharing the bottleneck link is also proposed and evaluated. While buffer dimensioning results in a small degradation in overall throughput, it aids connections with higher RTT and results in a better fairness coefficient.

Though simulation results are not presented for common scenarios in hybrid networks such as asymmetric channels, these have been the subject of extensive simulation in [31]. The results presented in [31] may be generalized to drop from front schemes such as PFD and are hence applicable to PFD performance.

Efficient implementation techniques for PFD are also discussed and Quasi-pushout PFD is introduced and evaluated in this context. Quasi-pushout PFD lends itself to high speed implementation while offering comparable fairness and goodput performance to the more rigorous PFD algorithm.

Some of our results indicate that an adaptive estimation of the fixed parameters in PFD, i.e. the drop probability and the threshold, is desirable. A level of performance tuning by parameter estimation is required for a scheme such as PFD. While the parameters chosen worked extremely well for most of the simulations in this chapter, it may be noted that under different conditions of buffer sizes and utilization, they may be less than optimal. Methods of dynamically estimating the drop probability and threshold based on the buffer size and utilization are the subject of future research.

Chapter 4

Buffer Management and Scheduling in Spoofing Gateways

4.1 Introduction

Internet over satellite has been widely investigated as a solution for providing the increasing bandwidth requirements of the end-user. In this context, Hughes Network Systems developed the Turbo^{TM} Internet product of DirecPC^{TM} in conjunction with the Center for Satellite and Hybrid Communication Networks at the University of Maryland [47, 48]. The asymmetric DirecPC^{TM} system uses receive-only satellite links for downstream data and uses a modem connection for the uplink data. Such a system enables delivery of high-bandwidth Internet access to the subscriber at up to ten times the speed achieved by conventional telephone line modems [47].

The authors in [48] note that this performance is bottlenecked by two factors.

First, the delay of the satellite link does not allow TCP to achieve high levels of throughput due to the time taken for the acknowledgments to return to the source. Second, the low bandwidth return path makes it expensive to acknowledge every TCP segment. The authors suggest the splitting of the end to end TCP connection into two segments, one from the application server to the hybrid gateway and the other across the satellite link. The hybrid gateway now emulates a TCP endsystem by acknowledging data from the application server. The latency of the satellite link is now hidden from the application server and does not affect throughput in the downstream direction. We refer to such connection splitting gateways as *spoofing gateways* in the rest of our discussion. Effects due to the low bandwidth return path are dealt with by selectively discarding acknowledgments in a "drop from front" manner. We note that schemes such as Probabilistic Fair Drop described in chapter 3 work well in such a scenario.

The DirecPC system is deployed at the edge of the network. [44] describes the deployment of connection splitting in a more generic architecture, in the middle of the network. The authors note that connection splitting proxies may employ a knowledge of underlying link characteristics to enhance performance, while the end to end semantics remain unaffected. Such an approach enables the interconnection of heterogeneous networks without an associated loss of performance, while the end to end protocols themselves remain unaffected by specific link level considerations.

[45] reports a simulation study of a similar spoofing system over satellite links. The satellite link is assumed to lie in the middle of the network, and the spoofing gateways operate with TCP enhancements for high performance such as window scaling, fast retransmit and recovery etc. The authors in [45] contend that a high performance satellite or hybrid system must implement a technique such as connection splitting to improve TCP performance. Their simulation results show increased link utilization and a decrease in the file download times observed by ftp clients.

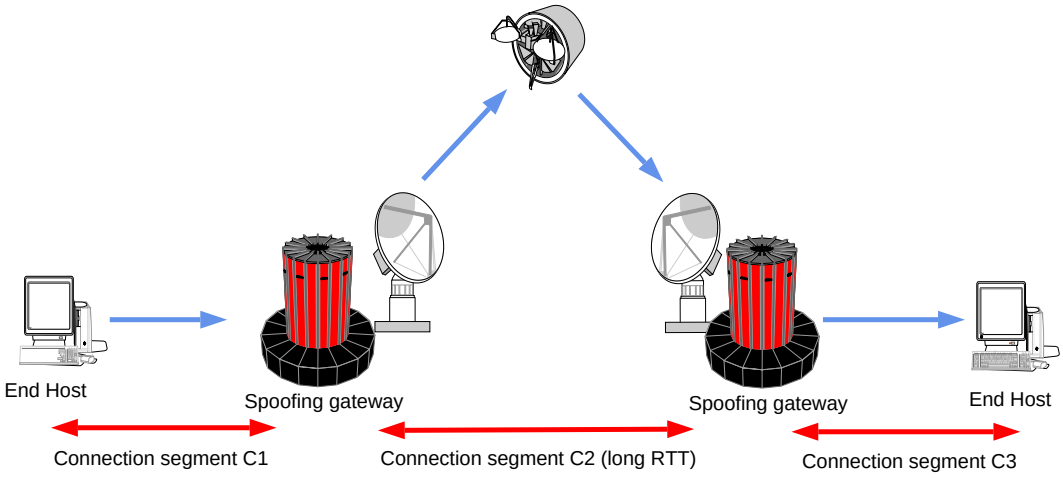

Figure 4.1: A generic connection splitting/spoofing overview. TCP connections are terminated in each segment and a new connection is established in the next segment.

Among the open problem areas related to spoofing are those of flow control and of queue management at the spoofing gateways. The nature of a spoofing gateway requires a large amount of buffering to keep the high-bandwidth satellite link full. In such a context, effective queue management as well as buffer dimensioning becomes vital. A generic spoofing architecture is depicted in Figure

4.1.

In preceding chapters, we outlined approaches to buffer management and scheduling on hybrid gateways in the Internet and stressed the need for the same. In this chapter, we investigate the specific class of spoofing gateways developed to enhance TCP/IP performance over satellite networks. Approaches to performing buffer management and scheduling are then evaluated in this scenario. We then propose an architecture for fair queueing and buffer management with these enhanced gateways and evaluate it by means of simulation.

4.2 Design Considerations in Spoofing Gateways

Several implementations of connection splitting gateways have been proposed in the literature [44] [46]. Our discussion closely follows the implementation of [44] and the simulation setup of [45]. We also consider the DirecPC^{TM} system setup in our discussions.

Spoofing or connection splitting gateways offer two primary advantages :

1. They isolate the satellite link from the terrestrial network (connection segment C2, in Figure 4.1). Thus, the satellite link may now run several TCP enhancements for high performance over high-bandwidth delay links while the end systems themselves need not implement these. Furthermore, the satellite link is not limited to using standardized protocols such as TCP. Implementations such as [44] report the use of increased values of the

initial window size over the satellite connection segment.

2. Spoofing gateways hide the latency of the satellite link from the terrestrial network. Since the first segment of the TCP connection is terminated at the satellite link (connection segment C1 in Figure 4.1), an end-host only sees the latency associated with the terrestrial segment C1. This reduction in observed round trip delay by the end host translates to an increase in throughput due to the "ACK-clocking" nature of TCP.

Several considerations must be taken into account when implementing a spoofing gateway. During connection establishment, a spoofing gateway must not establish connection segment C1 before it receives a response from the remote host. Thus a SYNACK must be returned to the source only after the receipt of an acknowledgment from the remote host. This introduces an unavoidable delay in connection setup which may be significant due to the long latency over satellite. Sequence numbers used on the connection segments C1 and C3 are also synchronized at the time of connection setup from information in the SYN segments. This enables recovery from data loss in the case of routing changes and so forth.

We now look closer into the data flow within a spoofing gateway. Our discussion draws upon the OPNET simulations described in [45].

Figure 4.2 depicts the data flow within a spoofing gateway for a unidirectional stream of data. The connection segments C1 and C2 are the same as in Figure

4.1. Incoming data on connection segment C1 traverses the link layer and is then queued at the FIFO queues of the IP layer. All TCP packets are now forwarded up to the TCP layer of the spoofing gateway. UDP datagrams and packets belonging to other protocols follow the normal flow of data in a traditional IP router, i.e. a routing decision is performed and these packets are forwarded to the appropriate outgoing interface.

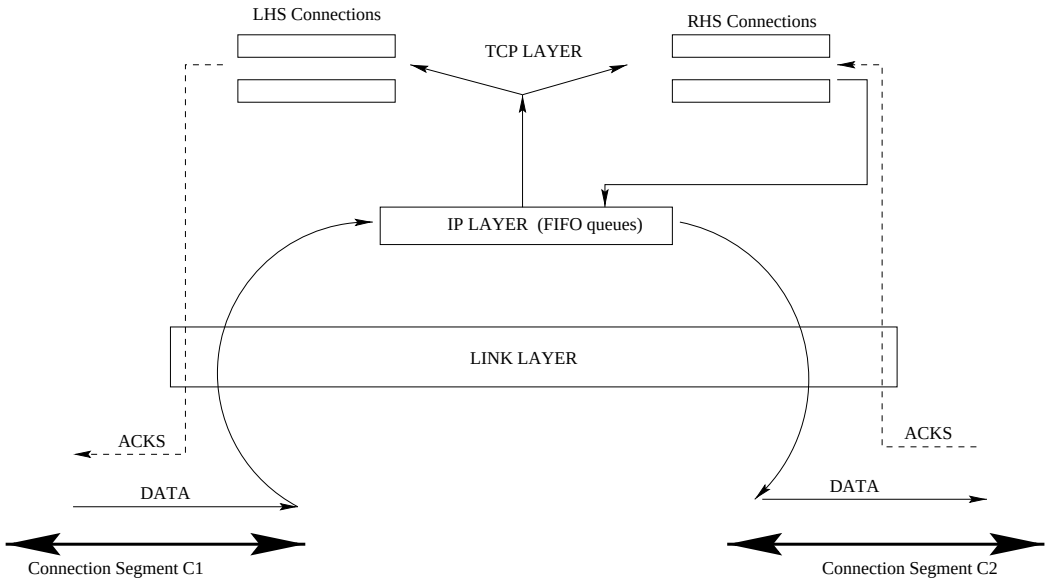

Figure 4.2: Data flow in a spoofing gateway. The solid lines represent data and the dashed lines represent acknowledgments.

At the TCP layer, the packets of each connection are copied and routed to two separate TCP connection processes. The first connection process , which we term the "left hand side" (LHS) connection terminates connection segment C1. The second connection process, the "right hand side" (RHS) connection initiates the connection segment C2. The LHS connection acknowledges the arriving TCP segment by sending back an acknowledgment to the initiator of the TCP

connection. The data received by the LHS is discarded after the acknowledgments are sent. This is accomplished by a periodic flush of the receive buffer. Note that this is necessary since there is no application process operating over the TCP layer.

The RHS connection buffers the data in the send buffer to be sent out on connection segment C2 until the offered window grows enough to enable it to transmit the received segment. The transmitted segment now passes through the FIFO queue in the IP layer again. The IP layer identifies it as a packet that has been processed by the spoofing gateway and forwards the IP datagram on the outgoing (satellite) interface.

There are several interesting artifacts about this data path. First, all TCP packets pass through the IP layer twice. An additional overhead is also introduced by the TCP layer processing. Thus, spoofing gateways introduce a significant amount of processing overhead at the cost of improved throughput performance.

Another point of note is the large buffering requirement of spoofing gateways. Since the LHS connection serves a much shorter round trip time segment than the RHS connection, the TCP connection in segment C1 is likely to build its window much faster than the TCP connection of segment C2. This rate mismatch needs to be absorbed in the spoofing gateway and such systems often have large buffers.

Flow control may also be implemented at the TCP layer in spoofing gateways. When acknowledgments are transmitted from the LHS connection on

connection segment C1, they reflect the available buffer from the RHS connection. This helps prevent the source host from transmitting at a rate that overflows the buffers of the RHS connection at the spoofing gateway.

We now take a closer look at the buffering of data in a spoofing gateway. Our intent is to motivate approaches to buffer management and scheduling in a spoofing system. Typically, packets are removed from the FIFO IP queues at the processing rate of the IP route server. This rate is unlikely to be a bottleneck in the system given the high processing rates of current day route servers. The buffering of data is predominantly limited to the RHS TCP send buffer queues. Thus on first glance, this appears to be the natural point for implementing some form of buffer management scheme. However, a closer observation shows that this is not the case. The TCP segments that occupy the buffers of the RHS connections have already been acknowledged. Thus, traditional buffer management schemes, which discard packets in anticipation of buffer overflow or to ensure a fair share, cannot be applied here. Furthermore, any buffer management scheme at the TCP layer would exclude packets from other protocols such as UDP. From our trace analysis of chapter 2, we know that in some scenarios this can be a considerable proportion of the traffic. An alternate approach is therefore called for.

4.3 An Architecture for Queue Management in Spoofing Gateways

The discussion in the previous section highlights several issues that we use to motivate our approach.

1. Spoofing gateways are likely to build up large buffers and hence intelligent management of these buffers is required.

2. Practically all the buffering in a spoofing system is at the RHS connections in the TCP layer. Traditional buffer management schemes, which discard packets, are not directly applicable here. Furthermore, any queueing strategy deployed here would not take into account packets from other protocols such as UDP.

Thus, spoofing gateways present a unique problem in queue management. From the preceding discussion, it is clear that any effective buffer management strategy would have to be applied below the TCP layer to account for non TCP flows. We consider the IP layer as a candidate for queue management. Several issues need to be resolved before effective buffer management strategies can be implemented at the IP layer. We consider each of them and propose solutions and work-arounds for them.

Multiple traversals of queueing system : In the spoofing architecture, TCP packets traverse the IP queues twice. If the IP layer is assumed to be a FIFO

queue, the TCP packets after being processed at the TCP layer, are placed at the tail of the queue a second time. Such a system is undesirable, but works in a "vanilla" spoofing system since there is not an appreciable amount of buffering at the IP layer. With buffer management and fair queueing strategies implemented at the IP layer such a data flow path is not desirable. Hence, we separate the IP queueing system into two subsystems. The first, which we term the fair queueing subsystem, enqueues packets that arrive from external interfaces and are yet to be processed. The second subsystem, the FIFO subsystem, enqueues processed TCP packets which arrive from the TCP layer.

Scheduling and queue management are performed at the first subsystem, while the second subsystem models a FIFO queue with no buffer mangement strategy. The validity of such an approach will be clear from the succeeding discussion. Figure 4.3 depicts the proposed architecture for queue management. We term this architecture as the *two server model* since a separate server is used for scheduling and another for IP route calculation.

What about TCP buffers ? The discussion in section 4.2 highlights that practically all the buffering in a spoofing system is at the TCP layer. This is because the bottleneck is likely to be the TCP sending rate which is window controlled. In such a scenario, buffer management at the IP layer is superfluous. Thus, any approach to buffer management at IP must first restrict the buffering in the system predominantly to the IP layer.

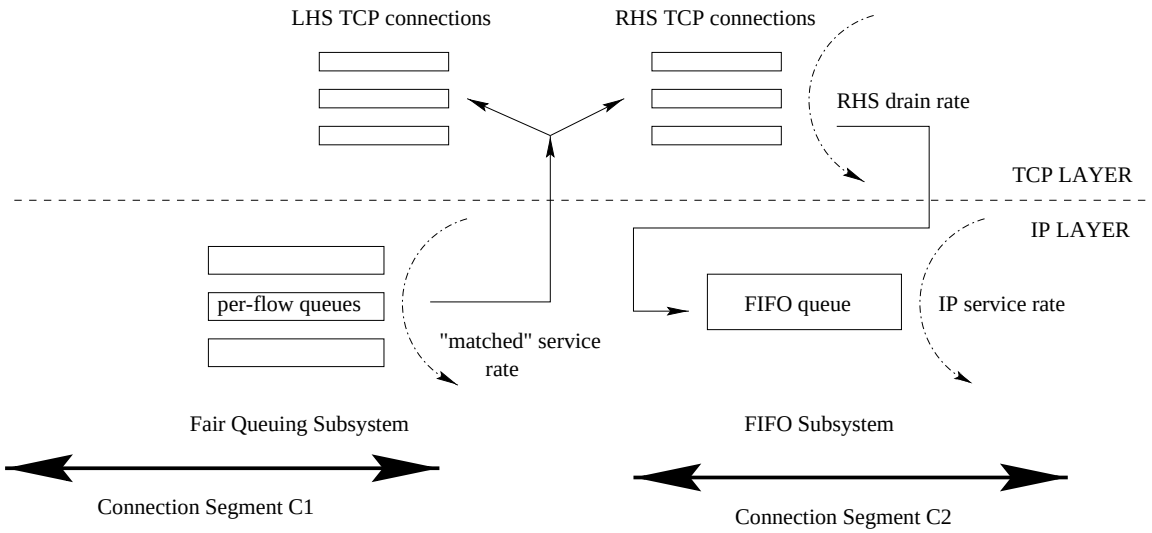

Figure 4.3: An architecture for queue management in a spoofing gateway. The solid lines represent the flow of data. The arcs represent the service rates of the respective queues.

To this end, we introduce the concept of a "rate-matching" server. We restrict the scheduling server in the fair queueing subsystem to serve the (per-flow) queues at a rate which *matches* the rate of the drain of the RHS TCP queues. The importance of rate-matching cannot be overemphasized. If the rate at which the per flow queues are served is much greater than the rate of drain of the TCP queues, then we cannot limit queueing effects to the IP queues. If the rate is too low, then starvation of the RHS TCP would result. An optimal rate for this server implies that packets arrive at the RHS TCP queues at a rate that minimizes buffering while not resulting in starvation. The determination of the optimal rate for this server is non-trivial and we discuss it further in a later section.

The rate-matching server now enables us to restrict the queueing effects to the IP fair queueing subsystem. Note that the presence of such a server also enables us to make do with FIFO queueing in the IP FIFO subsystem since a similar argument applies to this subsystem as to the RHS TCP queues.

The deployment of such a queueing architecture also enables a more tightly coupled feedback loop between adjacent connection segments. To understand this, we note that the received TCP segment for a connection is acknowledged after it is scheduled for service. Thus the propagation of acknowledgments on connection segment C1 is also scheduled under the auspices of the scheduling algorithm deployed. The vanilla spoofing architecture, even in the presence of fair queueing at the TCP layer, offers a much looser coupling between the adjacent connection segments. Backpressure, due to a connection attempting to grab more than its fair share of the resources, would thus percolate to the TCP source faster than in the vanilla case.

4.4 Simulation Study & Results

In this section, we study the buffering characteristics of the spoofing architecture by means of simulation. We deploy the two server architecture of section 4.3 for queue management and evaluate its effectiveness by means of simulation. Finally, we show that the proposed queue management architecture is effective in providing fair resource allocation on a hybrid gateway.

4.4.1 Simulation setup

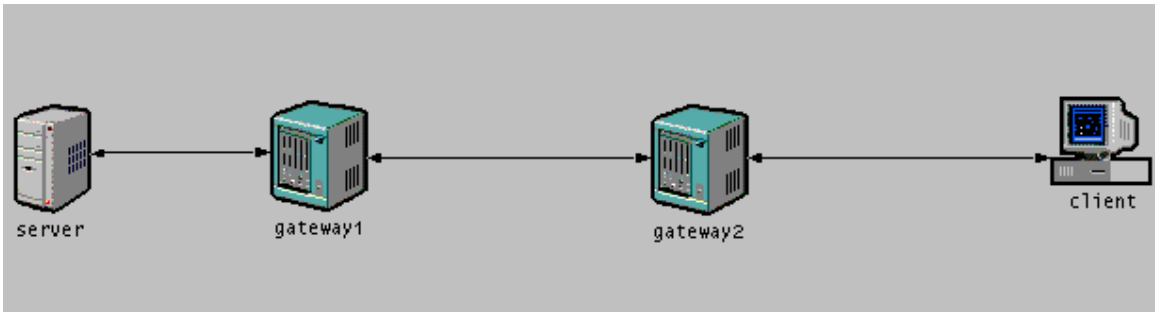

Figure 4.4: OPNET Simulation setup for the simulation study. The link between the two gateways is modeled as a satellite link with a round trip latency of 500 ms.

Our basic simulation setup parallels the one described by the authors in [45]. The OPNET network simulator [49] was used for all the simulations in this chapter. The network model used for most of the simulations is depicted in Figure 4.4.

The TCP layer on the hybrid gateways is configured with several standard TCP extensions for high performance. These include fast retransmit and recovery [38], window scaling [39] and Selective Acknowledgments (SACK) [50]. The gateways perform connection splitting and spoofing as described in [45]. The link between the hybrid gateways models a satellite link with a long round trip time (500 ms.). We do not account for satellite link error models and assume that these are dealt with at the link layer by an appropriate forward error correction scheme.

The server and the client are attached by 10BaseT Ethernet links to the gateways and the link delays are negligible in relation to the delay of the satellite link. The link rate of the satellite link is a configurable parameter.

4.4.2 Buffering characteristics of a spoofing system

To investigate the buffering characteristics of a spoofing system, we model file transfers involving large files from the server to the client in Figure 4.4. The parameters of interest are the utilization of the satellite link and buffering at the RHS TCP connections on the hybrid gateway. We focus our attention on gateway1 in Figure 4.4.

The link rate of the satellite link is set to T1 rate. The hybrid gateways are modeled with infinite send buffer capacity at the TCP layer for the purpose of simulation. Note that there is no flow control exercised by the spoofing gateways in our simulation setup. We tap the send buffer buildup in the RHS TCP layer of gateway1.

From the graphs of Figure 4.5, we see that the throughput on the satellite link is initially limited by the congestion window growth of the RHS TCP. The throughput graph flattens out when it hits the link bottleneck rate. A persistent buffer buildup may be observed in the RHS TCP send buffers even after the link throughput has stabilized. This may be attributed to the shorter RTT connection segment flooding the buffers of the hybrid gateway1. Since the long RTT connection segment operates with window scaling, the congestion window of TCP is not a bottleneck after the initial window buildup.

We now set the link rate of the satellite link to T3 rate. In this configuration, the satellite link rate is no longer the bottleneck in the system. We note from

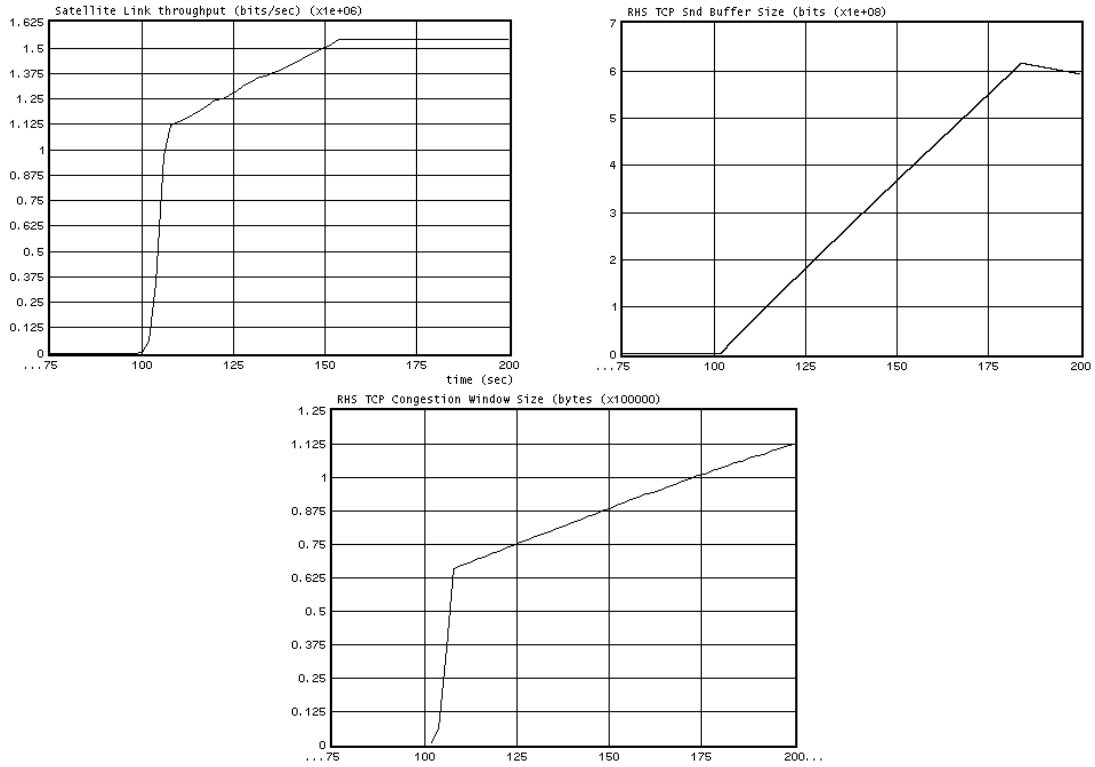

Figure 4.5: Satellite link throughput, RHS TCP send buffer size and RHS TCP congestion window evolution for a large file transfer with T1 link rates.

Figure 4.6 that the throughput curve flattens at a rate of about 1.6 Mbps with the gateway2 receive buffers set to 200 Kbytes. The bottleneck in this case was observed to be the receive buffers of the hybrid gateway terminating the long RTT connection segment. An increase in the receive buffers of the hybrid gateway results in a corresponding increase in the steady state throughput achieved over the spoofing system. This trend may be seen in the throughput curve with the receive buffer set to 400 Kbytes. In this case the steady state rate is about 3.4 Mbps. Corresponding increases in congestion window evolution and buffer drain rates can also be observed from the graphs of Figure 4.6.

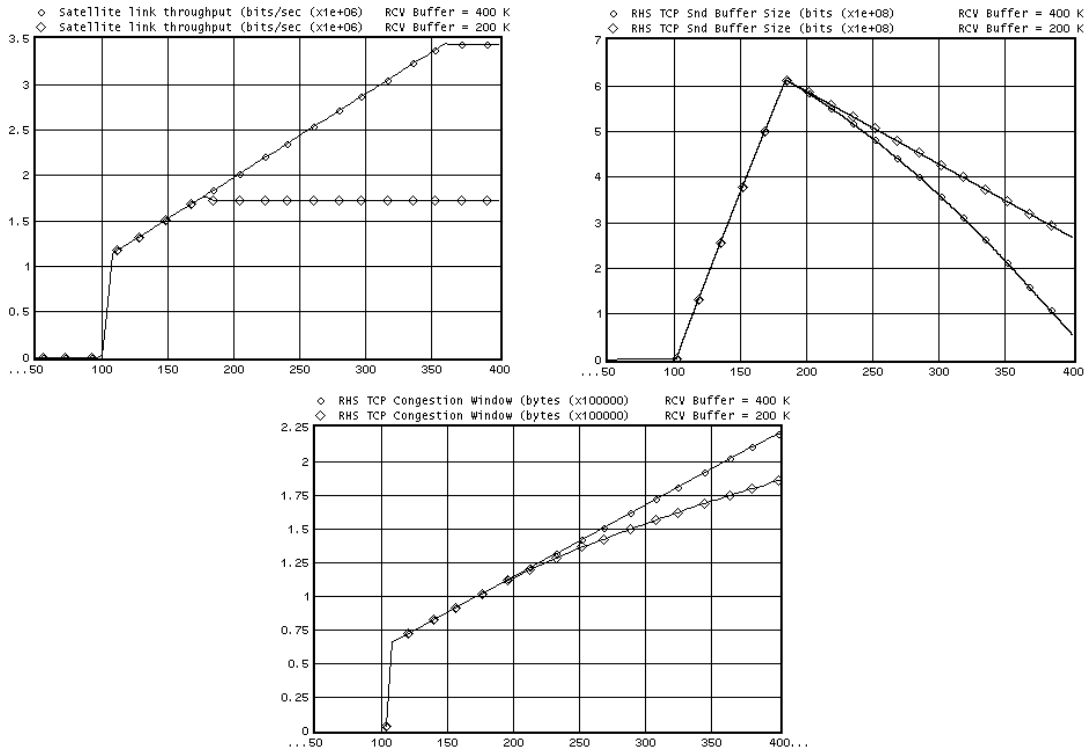

Figure 4.6: Satellite link throughput, RHS TCP send buffer size and RHS TCP congestion window evolution for a large file transfer with T3 link rates and varying RCV buffer on gateway2.

4.4.3 Effects of the two-server model

The OPNET TCP implementation maintains separate buffers for unsent data (the send buffer) and data that has been transmitted but has not yet been acknowledged (the unacknowledged buffer). The size of the unacknowledged buffer represents the current window size and is a function of the congestion window evolution and the bandwidth delay product of the outgoing link. The send buffer size is data that is buffered awaiting transmission. We are interested in limiting the size of the send buffer size on the spoofing gateways without a loss in throughput.

Our goal is therefore to maintain a backlog in the send buffer that is sufficient to feed the outgoing link without causing starvation and an associated loss of throughput. A drop in the size of the send buffer to zero is indicative of starvation while an unbounded increase in the size of the send buffer indicates unnecessary buffering of data. The latter trend is observed in the graphs of Figure 4.5 and Figure 4.6. In these scenarios, the bottlenecks arise from the link throughput and the receive buffers of the receiving node respectively, and the buffer buildup at the send buffers of the RHS TCP is wasteful and does not result in higher link utilization.

We now investigate the effects of deploying the two server model described in section 4.3. The service rate of the scheduler of the fair queueing subsystem is set to 500 packets/sec. We compare the obtained results with that of a vanilla spoofing system such as the one studied in the preceding section. The satellite link rate is set to T3 rates. The results are shown in Figure 4.7.

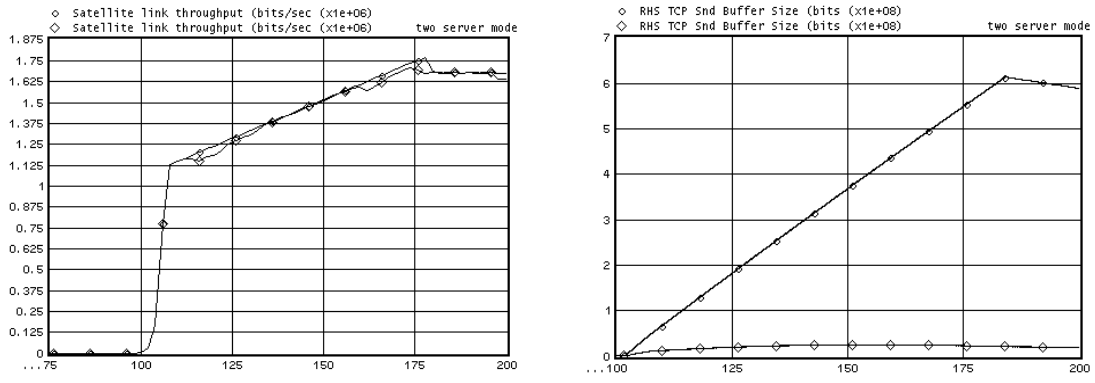

Figure 4.7: Comparison of satellite link throughput and RHS TCP send buffer size with and without the deployment of the two server mode.

The two server model has comparable throughput curves while successfully eliminating the wasteful buffering at the RHS TCP layer of the hybrid gateway to a large extent. This may be attributed to two factors :

1. The depressed service rate of the scheduler in the IP fair queueing subsystem exerts a form of implicit flow control on the sending TCP host, in this case the file server. Acknowledgments to the file server are delayed due to slower service rate of the scheduler. The scheduler rate is chosen such that it does not bottleneck the throughput of the spoofing system. In this case, the bottleneck is the size of the receive buffers of hybrid gateway 2. Thus, the delayed acknowledgments reduce the rate of the file server and thus reduce the unnecessary buffering at the RHS TCP layer.

2. This depressed rate also introduces increased buffering in the IP fair queueing subsystem. We compare the IP buffer sizes with and without the two server model. The results are shown in Figure 4.8. They indicate that though the buffering in the fair queueing subsystem is increased, the overall buffering in the spoofing system (TCP + IP) is still more than an order of magnitude lower. Thus, the effect of delayed acknowledgments dominates resulting in less wasteful buffering in the spoofing system.

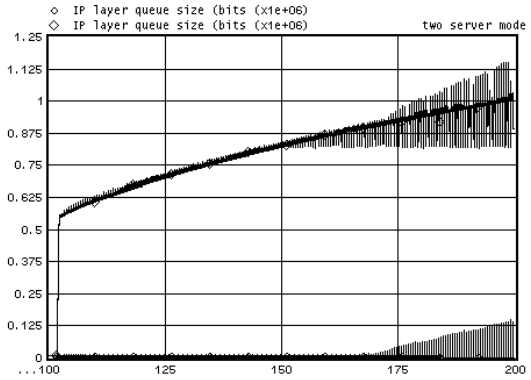

Figure 4.8: Comparison of IP layer buffering in hybrid gateway1 with and without the deployment of the two server mode.

4.4.4 Determination of an optimal rate

The results of section 4.4.3 demonstrate the effectiveness of the two server model in controlling the buffer space at the RHS TCP layer. We note that the a priori rate chosen in the previous section may be less than optimal and attempt to determine the optimal rate of the scheduling server by means of simulation. In this context, the optimal rate is one that offers minimum buffering at the RHS TCP layer with no loss of throughput. To this end, we run the simulations of the previous section with T3 satellite link rates and varying scheduler service rates.

The results are shown in Figure 4.9. For the lower scheduler rates of 400 and 450 packets/sec we note that the link throughput is bottlenecked by the rate of the scheduler and is lower than that observed with the vanilla spoofing system in Figure 4.6. In this case, the RHS TCP is starved after the initial burst in buffer size and the RHS TCP send buffers drop to zero. For the scheduler rate of 500 packets/sec some interesting behavior is observed. Initially the RHS TCP send

buffer builds up and the link throughput reaches the steady state value limited by the receive buffers of hybrid gateway 2. However, a decline in throughput is observed around 300 secs and the throughput settles down to a lower scheduler bottlenecked value.

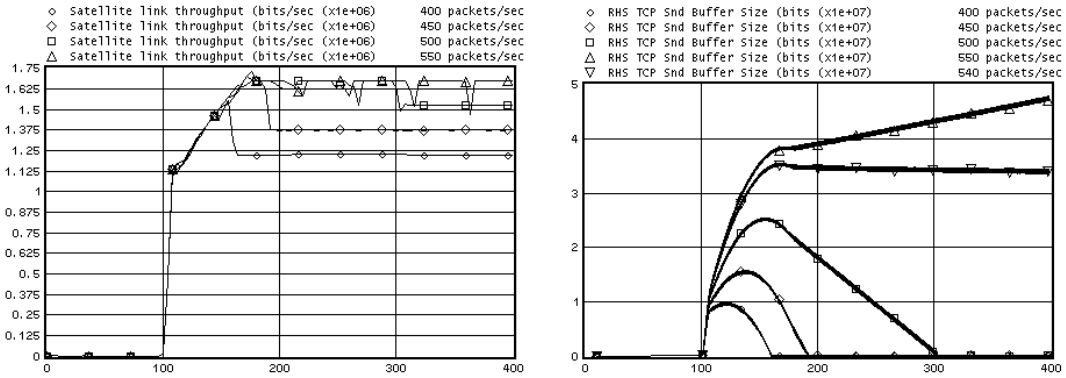

Figure 4.9: Determination of an optimal scheduler rate for the two server architecture.

To understand this behavior, we note that the scheduler serves the per flow queues at the IP layer not only for the data propagating in the forward direction but also for the TCP acknowledgments in the reverse direction. With increase in the window size and the data traffic, the load on the scheduler due to acknowledgments propagating in the reverse direction increases. This causes a decrease in the service rate available to the data in the forward direction. This decreased service rate now becomes a bottleneck in the system and the TCP send buffers begin to empty eventually resulting in a lower bottleneck rate for the system. The scheduler rate of 550 packets/sec is seen to be adequate to maintain the steady state link throughput.

The optimal rate appears to lie between 500 and 550 packets/sec for this simulation setup. Simulation estimates of the average serviced packet size at the IP fair queueing subsystem are approximately 3300-3400 bits/packet. Thus we see that this range represents the steady state throughput value of about 1.67 Mbps attained over the satellite link. Thus the simulation results corroborate our discussions of section 4.3. The optimal rate in this simulation setup is observed to be at a scheduler rate of 540 packets/sec. For this rate, the RHS TCP send buffer sizes achieves steady state with neither an unbounded increase nor a depletion to zero, as seen in Figure 4.9. The throughput graph for this scheduler rate is not depicted for the sake of clarity, but closely follows the 550 packets/sec curve.

4.4.5 Deployment of fair queueing

In this section, we evaluate the performance of fair queueing in a spoofing system with our proposed fair queueing architecture. We deploy the Start Time Fair Queueing (SFQ) [9] for its ease of high speed implementation. The buffer management technique employed is Probabilistic Fair Drop (PFD), since our results from chapter 3 indicate that it is ideal for such a scenario with long rtt connections over satellite.

TCP connections over satellite are fragile because of the long round trip time. The deployment of a spoofing architecture, while improving TCP performance dramatically, still renders them fragile in the presence of non adaptive flows such

as UDP and constant bit rate flows. To understand this, we note that in a vanilla spoofing system, TCP flows traverse the IP layer twice. On the second traversal, the packets are again enqueued at the tail of IP's FIFO queues and must await service a second time. Furthermore, in the absence of a fair queueing mechanism, non-adaptive flows are likely to grab a higher share of the resources at the bottlenecked system. The traffic analysis of chapter 2 indicates that the percentage of non TCP flows may not be negligible in several areas of the Internet. The rationale for the deployment of fair queueing at the IP layer follows from these observations. A spoofing system similar to the DirecPC^{TM} is employed for the simulations in this section.

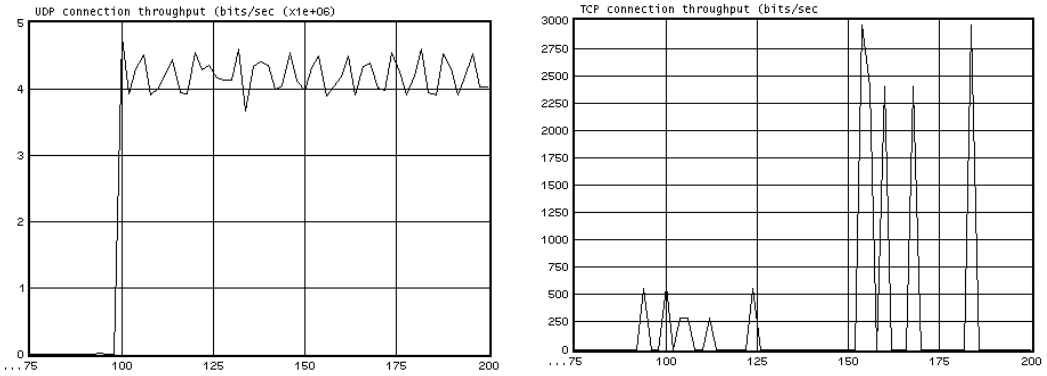

Figure 4.10: Throughput comparison of a TCP and UDP connection sharing a bottlenecked spoofing system.

We simulate an adaptive TCP connection and a non-adaptive UDP connection sharing a bottlenecked spoofing system. TCP extensions for improved performance such as window scaling and SACK etc. are deployed as before. The IP layer is modeled as a single FIFO queue. The results are shown in Figure 4.10.

The non adaptive connection manages to acquire most of the available link bandwidth and succeeds in shutting out the fragile TCP flow. The fluctuations in TCP throughput are caused by timeouts and retransmissions.

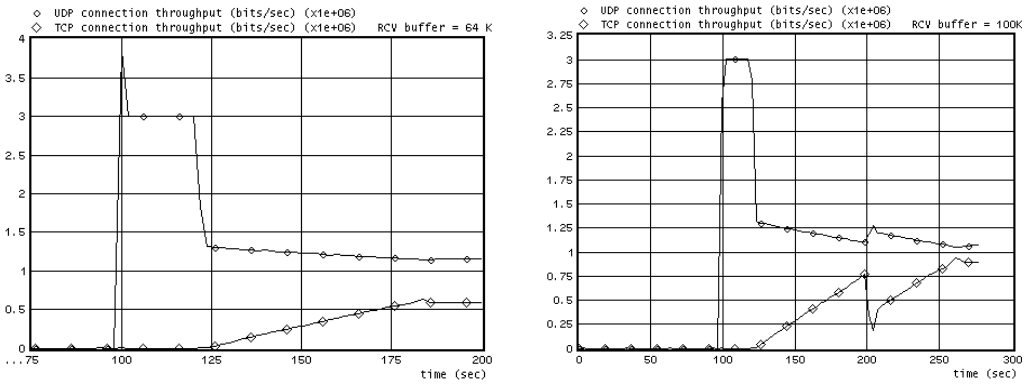

Figure 4.11: Throughput comparison of a TCP and UDP connection sharing a bottlenecked spoofing system in the presence of fair queueing.

The fair queueing architecture of section 4.3 is now deployed. The throughput comparison between the TCP and UDP connection is depicted in Figure 4.11. Initially, the UDP connection gains a large share of the resources while the long RTT TCP connection builds up its window. Once the TCP connection attains a large enough sending window, it obtains a fair share of the resources. In this case, the fragile TCP connection is protected by the deployment of fair queueing resulting in highly improved performance. The TCP connection in this case is bottlenecked only by the receive buffer size on the receiving node. An increase in the receive buffer size to 100K results in further performance improvement for the TCP connection as seen in Figure 4.11.

4.5 Conclusions and Future Work

In this chapter we investigated a special class of gateways proposed to enhance TCP performance over satellite networks, termed connection splitting or spoofing gateways. The unique characteristics of this system led us to propose a new architecture for fair resource allocation, which we term the two server architecture.

In section 4.4.2, we study the characteristics of a spoofing system and identify the various bottlenecks in a connection splitting/spoofing implementation. A large amount of wasteful buffering in the spoofing system due to the lack of flow control is uncovered. While our objective is not the deployment of effective flow control on a spoofing gateway, we show in section 4.4.3 that the wasteful buffering in a spoofing gateway may be reduced by more than an order of magnitude with no effect on the overall throughput performance of the system, with the help of the two server architecture. The determination of an optimal rate for the two server architecture is highlighted in the following section. The dynamic determination of an optimal rate in a spoofing system, for the purposes of auto-configuration, is an area for future research.

Finally, we evaluate the effectiveness of the two server architecture for fair queueing. Excellent results were observed in the case of adaptive and non-adaptive connections sharing a bottlenecked spoofing system. The deployment of the two server architecture coupled with fair queueing and buffer

management algorithms, succeeded in protecting fragile long RTT TCP flows from mis-behaved flows.

In this chapter, we have considered a spoofing system which does not deploy efficient flow control. Efforts are underway to develop efficient flow control algorithms for spoofing systems [51]. The interaction of such flow control algorithms with the architecture proposed in this chapter is an area for future research.

Chapter 5

Contributions of this work

In this chapter, we highlight the contributions of this dissertation.

Chapter 2 presents a framework within which the implementation overhead and hence feasibility of a fair queuing system can be evaluated. The various operations in a fair queuing system such as flow classification, scheduler tagging and issues such as the maintenance of flow state are analyzed in detail. A discussion on efficient algorithms to minimize implementation overhead uncovers the scalability associated with each of these operations.

This implementation overhead is further quantified by means of our trace analysis with the ASQG trace simulator. While flow classification studies abound in the literature, they either pertain to IP over ATM systems [21] [22] or are flow parsing studies, which do not utilize a processing element [52]. The latter class of studies, while providing interesting information about the nature of the traffic on a link, does not support analysis of the nature presented in chapter 2.

We analyze effects of flow aggregation and flow timeout values as a means of

improving the scalability of fair queuing implementations. Our results and analysis indicate that fair queuing schemes such as Deficit Round Robin (DRR) can be efficiently implemented with a small increase in overhead over FIFO queuing. The deployment of such systems appears to be feasible even with current demands of the Internet.

We contend that an analysis on the lines of chapter 2 coupled with a knowledge of system requirements is essential to determine the feasibility of deployment of fair queuing on a particular system. In this context, chapter 2 outlines an analysis methodology as well as presenting a set of sample results.

Our investigation of buffer management schemes in chapter 3, revealed several limitations of these schemes when used in conjunction with fair queuing over satellite networks. Probabilistic Fair Drop (PFD), our proposed buffer management scheme exhibits excellent performance in a combined throughput and fairness perspective. The use of soft thresholds and drop from front is combined with early detection of congestion in PFD to provide improved performance over satellite delay links.

Further extensions to PFD such as the buffer dimensioning based on an a priori knowledge of the individual round-trip times of TCP connections further alleviate the poor performance of long round-trip time connections. We note that in scenarios like the spoofing system of chapter 4 such round-trip time estimates are readily available from the split connections on the spoofing gateway.

The implementation overhead of a technique such as PFD, which requires the

scanning of multiple queues to determine the queue with the largest normalized buffer share, lead us to propose Quasi-pushout PFD. Quasi-pushout PFD provides an excellent approximation to PFD performance while lending itself to high-speed implementation.

The PFD algorithm, while offering excellent performance when deployed on hybrid gateways, requires performance tuning due to the fixed parameters in the algorithm. Dynamic parameter estimation for the PFD algorithm is the subject of future research.

The connection splitting/spoofing system [48] [44] is deployed to achieve improved performance over satellites. In chapter 4 we analyze the buffering dynamics of the spoofing system and present a novel architecture for queue management, the two server architecture. We uncover wasteful buffering in a spoofing system and show how it may be reduced by more than an order of magnitude with no impact on the throughput performance of the spoofing system.

The fragility of spoofed TCP connections in the presence of non-adaptive flows is one of the serious problems with spoofing gateways. The deployment of our two server architecture in conjunction with the fair queuing algorithms which are the subject of the earlier chapters, is shown to be effective in isolating non-adaptive flows and providing protection to the TCP connections.

While the two server architecture is not intended to replace flow control on spoofing gateways, it is intended to complement it. We believe that such an

architecture is essential for providing fair resource allocation and thereby robust TCP performance in a spoofing gateway. The importance of protecting adaptive flows from non-adaptive flows, is heightened by some of our trace analysis results of chapter 2, which show that a non-negligible proportion of flows on the Internet may be attributed to non-adaptive protocols such as UDP.

Indeed, such an argument may be applied in general to intermediate nodes on the Internet. Mechanisms for fair resource allocation and for optimizing traffic behavior are essential for the envisaged Internet of the future, with support for multiple classes of service, comprising the range from data through voice and video.

BIBLIOGRAPHY

[1] J. Postel (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification",*RFC 791*, September 1981.

[2] The ATM Forum, http://www.atmforum.com.

[3] The Differentiated Services Charter, http://www.ietf.org/html.charters/diffserv-charter.html.

[4] COMSAT Corporation's ATM Link Enhancer (ALE-2000), http://www.comsat.com/products.

[5] J. Nagle, "On packet switches with infinite storage", *IEEE Transactions on Communications*,vol. COM-35, no. 4, pp 435-438, April 1987.

[6] A.K. Parekh and R. Gallagher, "A generalized processor sharing approach to flow control - the single node case", *Proceedings of IEEE INFOCOM 1992*, vol. 2, pp 915-924, May 1992.

[7] A. Demers, S. Keshav and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm", *Internetworking: Research and Experience*, vol. 1, no. 1, pp 3-26, September 1990.

[8] S.J. Golestani, A Self-clocked Fair Queueing scheme for broadband applications", *Proceedings of IEEE INFOCOM 1994*, vol. 2, pp 636-646, April 1994.

[9] P. Goyal, H. Vin and H. Cheng, "Start-time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks", *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp 690-704, October 1997.

[10] L. Zhang, "VirtualClock: a new traffic control algorithm for packet switching networks", *ACM Transactions on Computer Systems*, vol. 9, pp 101-124, May 1991.

[11] D. Ferrari and D. Verma, "A scheme for real-time channel establishment in wide-area networks", *IEEE Journal on Selected Areas in Communications*, vol. 8, pp 368-179, April 1990.

[12] A. Varma and D. Stiliadis, "Hardware Implementations of Fair Queueing Algorithms for Asynchronous Transfer Mode Networks", *IEEE Communications Magazine*, pp 54-58, December 1997.

[13] S. Keshav, "An Engineering Approach to Computer Networking: ATM Networks, the Internet and the Telephone Network", Addison Wesley, 1997.

[14] S. Keshav and R. Sharma,"Issues and Trends in Router Design,*IEEE Communications Magazine*, May 1998.

[15] D. Decasper, M. Waldvogel, Z. Dittia, H. Adiseshu, G. Parulkar and B. Plattner,"Crossbow - A Toolkit for Integrated Services over Cell Switched IPv6",*Proceedings of the IEEE ATM'97 workshop.*

[16] P. McKenney, "Stochastic Fairness Queueing", *Proceedings of IEEE INFOCOM 1990*, vol. 2, pp 733-740, June 1990.

[17] M. Shreedhar and G. Varghese, "Efficient Fair Queueing using Deficit Round Robin", *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp 375-385, June 1996.

[18] S. Keshav, "On the efficient implementation of fair queueing", *Internetworking: Research and Experience*,vol. 1, pp 157-173, September 1991.

[19] T. Cormen, C. Leiserson and R. Rivest, "Introduction to Algorithms", MIT Press/McGraw-Hill, 1990.

[20] R. Jain, "A Comparison of Hashing Schemes for Address Lookup in Computer Networks," *IEEE Transactions on Communications*, vol. 40, no. 3, pp 1570-1573, October 1992.

[21] S. Lin and N. McKeown, "A Simulation study of IP Switching", *Proceedings of ACM SIGCOMM 1997*, vol. 27, no. 4, pp 15-24, October 1997.

[22] H. Che and S. Li, "Adaptive Resource Management for Flow- Based IP/ATM Hybrid Switching Systems", *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp 544-557, October 1998.

[23] Sanitize scripts from the Internet Traffic Archive, available for download at http://ita.ee.lbl.gov/html/contrib/sanitize.html.

[24] Traces in the Internet Traffic Archive, http://ita.ee.lbl.gov/html/traces.html.

[25] National Laboratory for Applied Network Research(NLANR) network traffic traces, http://moat.nlanr.net/Traces/.

[26] R. Vaidyanathan, R. Srinivasan and J.S. Baras, "Buffer Management strategies for per flow queueing hybrid satellite gateways", *paper in preparation.*

[27] V. Jacobson, "Presentations to the IETF Performance and Congestion Control Group",1989.

[28] E. Hashem,"Random Drop Congestion Control", M.S. Thesis, Massachusetts Institute of Technology, Department of Computer Science, 1990.

[29] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp 397-413, August 1993.

[30] D. Lin and R. Morris, "Dynamics of random early detection", *Proceedings of ACM SIGCOMM 1997*, vol. 27, no. 4, pp 127-137, October 1997.

[31] B. Suter, T.V. Lakshman, D. Stiliadis and A. K. Choudhury, "Design considerations for supporting TCP with per-flow queueing", *Proceedings of IEEE INFOCOM 1998*, vol. 1, pp 299-306, April 1998.

[32] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, pp. 115-156, September 1992.

[33] V. Jacobson,"Congestion Avoidance and Control", *Proceedings of ACM SIGCOMM 1988*, vol. 18, no. 4, pp 314-329, August 1988.

[34] A. Mankin and K.K. Ramakrishnan, "Gateway Congestion Control Survey",*RFC 1254*, August 1991.

[35] T.V. Lakshman, A. Nierhardt and T.J. Ott, "The Drop from front strategy in TCP over ATM and its Interworking with other Control Features", *Proceedings of IEEE INFOCOM 1996*, vol. 3, pp. 1242-1250, March 1996.

[36] T.V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss", *IEEE/ACM Transactions on Networking*, pp. 336-350, June 1997.

[37] J. Postel(ed.), "Transmission Control Protocol - DARPA Internet Program Protocol Specification", *RFC 793*, September 1981.

[38] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit and Recovery Algorithms", *RFC 2001*, January 1997.

[39] V. Jacobson, "TCP Extensions for High Performance", *RFC 1323*, May 1992.

[40] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks", *Computer Networks and ISDN Systems*, vol. 17, no. 5, pp 1-14, 1989.

[41] OPNET Modeler/Radio release 4.0A, http://www.mil3.com.

[42] J. Padhye, V. Firoiu, D. Towsley and J. Kurose,"Modeling TCP Throughput : A Simple Model and its Empirical Validation", *Proceedings of ACM SIGCOMM 1998*, vol. 28, no. 4, pp 303-314, October 1998.

[43] Y.S. Lin and C.B. Shung, "Quasi-pushout cell discarding", *IEEE Communications Letters*, vol. 1, no. 5, pp. 146-148.

[44] N.P. Butts, V.G. Bharadwaj and J.S. Baras, "Internet Service via Broadband Satellite Networks", *Multimedia Systems and Applications: Proceedings of SPIE*, pp. 169-180, February 1999.

[45] M. Karir, M. Liu, B.A. Barrett, J.S. Baras, "A Simulation Study of Enhanced TCP/IP Gateways for Broadband Internet over Satellite", *submitted to OPNETWORK '99*, September 1999.

[46] H. Balakrishnan, V.N. Padmanabhan, S. Seshan and R.H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links",*Proceedings of ACM SIGCOMM 1996*, vol. 26, no. 4, pp 256-269, October 1996.

[47] A.D. Falk, "A System Design for a Hybrid Network Data Communications Terminal Using Asymmetric TCP/IP to Support Internet Applications", *M.S. Thesis*, University of Maryland, 1994.

[48] V. Arora, N. Suphasindhu, J.S. Baras, D. Dillon, "Asymmetric Internet Access over Satellite-Terrestrial Networks", *Proceedings of the AIAA: 16th International Communications Satellite Systems Conference and Exhibit, Part 1*, pp 476-482, February 1996.

[49] OPNET Modeler/Radio release 5.1C, http://www.mil3.com.

[50] S. Floyd, M. Mathis, J. Mahdavi and A. Romanov, "TCP Selective Acknowledgment options",*RFC 2018*, October 1996.

[51] V.G. Bharadwaj, "Improving TCP Performance over Satellite Links",*M.S. Thesis*, University of Maryland, August 1999.

[52] NLANR Tutorial, Flow counts as a function of flow granularity, http://www.nlanr.net/NA/Learn/aggregation.html.