Copy of final edition

ACHIEVING END-TO-END DELAY BOUNDS IN A REAL-TIME STATUS DISSEMINATION NETWORK

By

JOEL NORMAN HELKEY

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electricial Engineering and Computer Science

MAY 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of JOEL NORMAN HELKEY find it satisfactory and recommend that it be accepted.

____

Chair

____

____

unsigned copy

Acknowledgment

I owe the successful completion of this thesis to the excellent guidance that I have received from my thesis chair, Dr. Carl Hauser. His encouragement and support have been invaluable to me throughout this process. I would also like to thank both Dr. David Bakken and Dr. Murali Medidi for serving on my thesis committee.

My research at Washington State University was funded in part by a Research Assistant position, granted upon my arrival by the School of Electrical Engineering and Computer Science, for which I am most grateful. I am also very appreciative of the helpful EECS staff, especially Ruby Young, for guiding me through the many bureaucratic hurdles I faced on my way to graduation.

I acknowledge all the members of the GridStat research group and especially want to thank Venkata Irava and Kjell Harald Gjermundrød for familiarizing me with the Java implementation of GridStat. I extend thanks to John Fisler, a friend from my days working at Lattice Semiconductor Corporation, for his support of this research by sharing his technical knowledge and engaging in many thought-provoking discussions.

This thesis would not have been possible without the unconditional love and support from my wife, Arlene. I am grateful for the encouragement of my parents, George and Carole, my sister Kirsten, my brother Roger and all the members of my extended family for their wholehearted support. They make my life so much more meaningful, and I am blessed to have them all in my life.

Publications

Carl H. Hauser, David E. Bakken, Ioanna Dionysiou, K. Harald Gjermundrød, Venkata S. Irava, **Joel Helkey**, and Anjan Bose. "Security, trust and QoS in next-generation control and communication for large power systems". International Journal of Critical Infrastructures, to appear 2007.

ACHIEVING END-TO-END DELAY BOUNDS IN
A REAL-TIME STATUS DISSEMINATION
NETWORK

Abstract

by Joel Norman Helkey, M.S.
Washington State University
May 2007

Chair: Carl Hauser

Status dissemination networks are used in many critical infrastructures. Most notably they can be found in electric utilities, oil and gas distribution systems, and water distribution systems. The GridStat project is a status dissemination middleware framework that is being developed for the electric power grid. GridStat is an event-based application framework built on a publisher-subscriber paradigm. It has admission control for new traffic flows and uses a multicast routing protocol in the routers. When the traffic travels solely over a dedicated network, GridStat will provide the following Quality of Service (QoS) properties: reliability and real-time performance.

This thesis focuses on providing real-time performance, sometimes referred to as timeliness, for GridStat on a dedicated network. An analysis of current methods in bounding end-to-end packet delay based on various Guaranteed Rate (GR) scheduling algorithms is performed, with the Delay Earliest Due Date (Delay-EDD) algorithm being proposed as the most appropriate one to be used in this case. GR scheduling algorithms can give a deadline, or delay guarantee, by which a packet of a flow will be transmitted. GR scheduling algorithms also provide a delay guarantee for a flow that is independent of the behavior of other flows in the network.

The result of using dedicated network links, admission control, resource reservation, and a guaranteed rate scheduling algorithm in the routers of a status dissemination network, is the ability to provide deterministic guarantees on end-to-end packet delay. Real-time performance for the GridStat framework is thereby achieved.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

The aim of this thesis is to provide real-time performance for the GridStat framework, [BBH+02, GDB+03, HBB05]. The fundamental issue in real-time performance is how to create a mechanism within the network that will provide guaranteed packet end-to-end delay bounds. Bounded end-to-end delay for all packets cannot be satisfied by current networks that only support best effort delivery and use a simple scheduling algorithm, such as First In First Out (FIFO), in the routers.

GridStat is a middleware framework that provides a status dissemination service for the electric power grid and is designed to overcome the limitations of the current electric power grid communication infrastructure. Unlike many industries where the matching of supply and demand are not so time dependent, the electric power industry is one where instantaneously matching supply and demand is critical. Failure in balancing them can result in a partial or complete shutdown of the grid system, [MA99]. Operators maintain proper balance by getting grid status information in a timely manner. Therefore, real-time status dissemination for the electric power grid is a critical component for maintaining reliable power delivery and avoiding blackouts.

Due to the critical nature of status dissemination for the electric power industry, it is more appropriate to characterize GridStat as requiring hard real-time as opposed to soft real-time. Hence, a deterministic technique or approach should be used to guarantee bounded end-to-end delay in GridStat.

1.1 Motivating Example

The following is a motivating example for providing end-to-end delay bounds for grid status information. Consider a electric power substation that is networked to a data center using T1 lines (1.5 Mbps) for the purpose of delivering status information. At the substation, it is producing 5 Phasor Measurement Unit (PMU) flows (a flow is a stream of packets being transmitted from the source) and 50 status variable (Pub) flows. The PMU flows are producing 5 PMU measurements 60 times per second and the Pub flows are producing 50 status variables each one time per second.

Further, assume that the PMU measurements are a part of a Special Protection Scheme, [TBVB05], used to protect the electric power grid. And each of the packets from the PMU flows are required to have a maximum end-to-end delay of 10ms.

For the five PMU traffic flows, the packet size is 72 bytes and the interval between packets is 16.67ms. For the 50 Pub traffic flows, packet size is 50 bytes and the interval between packets is 1000ms. In a worst case situation, over a network of routers using FIFO scheduling, a PMU measurement could be delayed by as much as 15.6ms. This would exceed the required maximum end-to-end delay by 5.6ms.

Now change the scheduling algorithm in the routers from FIFO to one where the scheduling is based on packet deadlines. Upon arrival at the router, each packet gets stamped with a deadline and is guaranteed to be sent on the outgoing link before the deadline expires. Under these conditions, all five PMU measurements could be scheduled to meet a 1 ms deadline.

1.2 Thesis Contributions

The aim of this thesis is to provide real-time performance for GridStat. Providing real-time performance for a large scale status dissemination network is challenging. First of all, if there was a need to exclusively use the Internet which has no limits on traffic admission, it would not be possible to provide deterministic bounded delay guarantees. However, by utilizing a dedicated network, appropriate scheduling, and traffic admission control in the GridStat framework, the problem becomes amenable to a workable solution. Under these constraints, for networks that employ scheduling algorithms belonging to the class of algorithms known as Guaranteed Rate (GR), [GLV97], it is possible to determine an upper bound on end-to-end delay.

To obtain delay guarantees, [GLV97], the routers on the path reserve a rate for a flow of packets and use a rate-based scheduling algorithm. Based on the rate reservation, many scheduling algorithms can guarantee a deadline, or delay guarantee, by which a packet of a flow will be sent on the outgoing link. The class of rate-based scheduling algorithms that provide such delay guarantees is known as Guaranteed Rate (GR) scheduling algorithms. GR scheduling algorithms also provide a delay guarantee for a flow that is independent of the behavior of other flows in the network.

The primary contributions of this thesis are:

- An explanation of why FIFO or Priority scheduling algorithms are not sufficient to provide bounded end-to-end delay in a large scale status dissemination network.
- The identification and analysis of existing guaranteed rate scheduling algorithms. Selection of Delay-Earliest Due Date (Delay-EDD) as the most appropriate guaranteed rate scheduling algorithm for GridStat.
- A basic conceptional structure for bounded end-to-end delay within GridStat is outlined. It is a comprehensive solution and includes the necessary elements of dedicated network, admission control, resource reservation, and guaranteed rate scheduling algorithm (Delay-EDD) in the routers.
- The implementation of the Delay-EDD scheduling algorithm in a GridStat prototype.

Included is a detailed explanation of how the status router java code was changed from FIFO to Delay-EDD.

- Experimental evaluation of Delay-EDD as implemented in GridStat. The results demonstate that a Delay-EDD router can deliver bounded end-to-end delay in cases where a FIFO router cannot.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 explains the background necessary to understand scheduling algorithms. A review of the related research for delay guarantees is presented in chapter 3. Chapter 4 covers a conceptional structure for delay guarantees in GridStat. Chapter 5 introduces a guaranteed rate scheduling algorithm (Delay-EDD) implemented in the GridStat status routers. The experimental evaluation and analysis of results are presented in chapter 6. The thesis ends with a conclusion and presentation of potential future work in chapter 7.

Chapter 2
Background

This chapter explains the concepts of Quality of Service (QoS) and Real-Time. It presents a specific Status Dissemination Network called GridStat, the middleware framework used for the experiments conducted in this thesis. Then it is shown how an event or message travels from source to destination within this framework. The last section of this chapter discusses what level of service applies to GridStat.

Throughout this thesis Internet Protocol networking and packet-switching is assumed, with traffic being statically routed over a Wide Area Network. A Wide Area Network (WAN) is a data communications network that covers a relatively broad geographic area, up to worldwide locations. In contrast to a local area network, a WAN is not contained within a limited geographical location. A WAN could be assembled from transmission lines leased from a commercial telecommunication carrier or assembled from dedicated transmission lines constructed by the organization itself.

The unit of data transmission at the network level is the packet. The sequence of packets transmitted by a source is refered to as a *flow*. Networks consist of two components: transmission lines (links) and switching elements (nodes). Transmission lines transfer a stream of packets from one end of the line to the other at a certain rate and with a fixed propagation time. Typical media used for transmission lines are copper coaxial cable, fiber optic, and microwave or radio wireless links.

Switching elements are specialized computers that connect transmission lines. The functions performed by switches are to multiplex and demultiplex packets belonging to different computer to computer flows, and to determine the link(s) along which to forward any given packet. This task is essential because a given transmission line will usually be shared by several concurrent sessions between different computers. These switching elements have had different names in the past, but now *router* is most commonly used.

Packet-switching is the basis for the Internet Protocol (IP). Since there are many possible paths through the network providing connections for many computers, packet-switching networks provide any computer to any computer connections. IP solves the problem of connecting different networks together and packet-switching improves link utilization and network throughput.

In packet-switching, the packets are sent, to the nearest router, which looks up the destination address and forwards it to the next hop. This process is repeated until the packet reaches its destination. This forwarding mechanism is called store-and-forward because IP packets are completely received and stored in the router while being processed, and then transmitted. Also, packets may need to be buffered in a queue at the router while waiting for an outgoing link.

2.1 Motivation for QoS

Local and wide area computer communication networks are now ordinarily packet-switched networks, replacing the earlier telephone-based circuit-switched networks.

In circuit-switched networks, a dedicated path known as a circuit is established between the communication end points and resources needed along this path (for example buffers and bandwidth) are reserved for the entire duration of the communication session. The network is responsible for allocating sufficient resources to allow the sender to transmit data as a continuous data flow, only limited by its peak transmission rate. The Public Switched Telephone Network (PSTN) is an example of a circuit-switched network.

Circuit-switching is not very efficient because the dedicated circuits are idle during times when the sender and receiver are not exchanging any messages. This results in poor utilization of the link bandwidths.

To address this problem, packet-switched networks were introduced as an alternative to circuit-switching. This technique allows packets from different sources to share the links resulting in a more efficient utilization of the link capacities. However, as a consequence of using a packet-switched network and finite buffer space, a new way of dropping packets is introduced. If one of the links is congested because multiple packets need to be transmitted over the same link at the same time, then one of the packets is chosen for transmission and the rest have to be stored in a buffer. As a result, buffering is required to absorb traffic bursts and prevent possible packet losses due to limited buffer space.

There are basically two motivations for implementing QoS capability in a network. The most obvious motivator is that different applications require different service from the network in order to function properly. Many data oriented applications can live with best effort networking, no QoS other than almost reliable delivery. On the other hand, real time applications such as VoIP or video-conferencing generally have QoS requirements. For example, to support an application that carries voice traffic as a 64 Kbps stream, the network must provide a minimum of 64 Kbps bandwidth on the path from end-to-end. In addition, VoIP typically requires 200 ms or less one way delay. Consequently, this application and others of its kind, demand a guarantee of some minimum level of service from the network.

Another motivation behind enabling QoS guarantees is to achieve service differentiation for different flows belonging to different users of the network. This second motivator is an economic one: if an ISP, for example, is able to provide quality of service differentiation, it can charge the customers differently depending on what level of service they are willing to pay for. On the Internet, traffic may consist of real-time traffic for applications such as VoIP or multimedia. A second class of service may be for applications such as transaction processing that require reliable data delivery. Another class of service to be carried on the network is best-effort traffic from applications such as file transfer and e-mail.

Different applications have varying traffic characteristics with different requirements, and rely on the ability of the network to provide QoS guarantees with respect to several measures, such as throughput, packet loss, delay, and jitter. However, network resources such as link bandwidth and buffers are shared by multiple users or services, some or all of which may try to access a resource simultaneously. Resource contention arises because of this sharing. A QoS mechanism is needed to efficiently allocate and manage limited network resources among competing users.

2.2 Quality of Service

QoS refers to network performance measures for a flow (stream of packets from a source to a destination) such as throughput, packet loss, delay, and jitter, as seen by users and applications. *Throughput* is effective data transfer rate and is measured in bits per second. *Packet loss* is the percentage of packets lost or dropped in the network over a given time period. *Delay* (or latency) is the time taken by the data packet to travel from source to destination, typically measured in seconds. And finally, *jitter* is the variation in the packet arrival times at the destination, measured in seconds.

QoS is a measure of how quickly and reliably the network does its job of delivering data from source to destination. In simpler terms, QoS could be thought of as providing a predictable data delivery service, [Puz02].

The end-to-end delay for a packet is the summation of transmission, processing and queuing delays in routers; propagation delays in the links; and end-system processing delays along a path from source to destination, [KR05].

The *processing delay* is the time required to examine the packet header and assign the packet to an outgoing link queue. For example, the processing delay in a router could include timestamping a packet upon arrival, calculating the deadline to attach to the packet, and then placing the packet in the right place in a sorted queue.

The *queuing delay* is the time from when the packet is assigned to a queue for transmission and the time it starts getting transmitted onto the link. The length of the queue will depend on the number of packets that have already arrived and are waiting for transmission across the link. If the queue is empty, then the queuing delay will be zero. However, if there are many packets already in the queue, then the queuing delay could be relatively long.

The *transmission delay* is the amount of time required to transmit all of the packet's bits into the link. $Delay_{trans} = \frac{L}{R}$, where L is the length of the packet in bits and R is the transmission rate of the link in bits per second.

The *propagation delay* is the amount of time it takes for a bit to travel on the link. The bit propagates at the propagation speed of the link, which depends on the physical medium of the link (for example, fiber optics, copper wire, etc.). $Delay_{prop} = \frac{d}{s}$, where d is the length of the physical link and s is the propagation speed in the medium. For example, with copper wire as the medium, $s \approx 2.3 \times 10^8\ meters/sec.$

Depending on the application, QoS needs can vary and different levels of service could be required. While best effort can work at times, video and audio are examples of applications that typically need QoS guarantees. Delay and jitter must be upper-bounded to ensure real-time delivery. In general, the QoS guarantee required for an application could be best effort, deterministic, or statistical.

A best effort level of service is basic service without explicit QoS guarantees. This level of service is best characterized by a network of routers with FIFO queues, which have no differentiation between flows.

Deterministic QoS guarantees provide each packet on the traffic flow with an absolute guarantee on the worst-case end to end delay and are based on a worst case analysis of the network delays at each router on the path. One approach uses what is called network calculus, [Cru91a, Cru91b], for characterizing the data flow through packet-switching networks. By analyzing the potential burstiness of traffic flows, network calculus identifies the possible worst case packet delay queuing up at various points in a network. This was an important work in the field because prior mathematical analyses of queuing delays were mostly limited to statistical approaches that addressed only a single queue.

In their paper "Determining End-to-End Delay Bounds in Heterogeneous Networks", Goyal et al., [GLV97], have extended the work of Cruz. They define a class of Guaranteed Rate (GR) scheduling algorithms and use it to determine an upper bound on end-to-end delay for a variety of sources. They observe that the end-to-end delay of a packet depends on the source traffic characteristics and the scheduling algorithm at the network routers or switches.

Statistical QoS guarantees allow a fraction of the traffic to violate QoS specifications. A statistical delay bound is defined as, [Fer90]:

$$Prob(D_i \leq D_{max}) \geq Z_{min},$$

where D_i is the delay with which the i-th packet sent by the source is delivered to the destination and D_{max} is the upper bounded delay as specified by the source at admission time and Z_{min} is the lower bound of the probability for successful and timely packet delivery. Lack of a measurement time interval specification is the main problem with this equation. No matter how many packets on a flow have been delayed beyond their bound, it is always possible for the router to correct the situation in the future and meet the given statistical requirements. A possibly more verifiable definition for a statistical bound would be a fractional one. For example, a statistical bound could be specified as follows: out of any 100 consecutive packets on the flow, no less than 97 will be on time.

While the statistical approach is not explored in this thesis, it is not completely out of the question for use in a Status Dissemination Network in the general case. It just depends on how critical timely delivery is for the application of interest. For an application that can tolerate a fraction of lost or delayed packets, this can be a reasonable alternative to the deterministic approach because it can lead to a higher utilization of the network links. Unfortunately, the statistical approach opens up the possibility of important messages being delayed to the point where they miss their deadline or are dropped due to a buffer overflow.

2.3 Real-time

A real-time system is a system where the correctness of the system depends not only upon the logical result of the computation, but also on the time at which the results are produced, [Mok83, Sta88]. This dependence on times is frequently inherent in the problem. An example is a system for updating an airplane cockpit display, where the system is only correct if the displayed values are less than one hundred milliseconds old.

It is common to classify real-time systems into hard or soft real-time systems. Hard real-time systems are those in which any failure to satisfy a timing constraint by any amount is considered a system failure. An example of a hard real-time system might be the flight software of a spacecraft. Firing a motor even a millisecond late could result in failure to achieve orbit on a long space voyage.

Soft real-time systems are those where satisfying timing constraints are important, but the system will still function correctly if missed deadlines only lead to less throughput or an acceptable reduced QoS. An example of a soft real-time system is a multimedia application in which there is a relatively high tolerance for missing deadlines related to the transferring of sound or video. Such failures might result in temporary degradation of quality, but the presentation itself remains largely intact. Users of such an application are often willing to tolerate a few dropped sound bites or video frames, as long as the glitches occur rarely and are of relatively short lengths of time.

2.4 Status Dissemination Network

GridStat, [BBH+02, GDB+03, HBB05], is a status dissemination middleware framework that is being developed for the electric power grid. It is an event-based application framework built on publisher-subscriber communication, [OPSS93], which is a variant of a generative communication model, [CG89]. Middleware, see figure 2.1, are the services layered between the applications and an operating system that provide specialized services and interoperability between distributed applications.

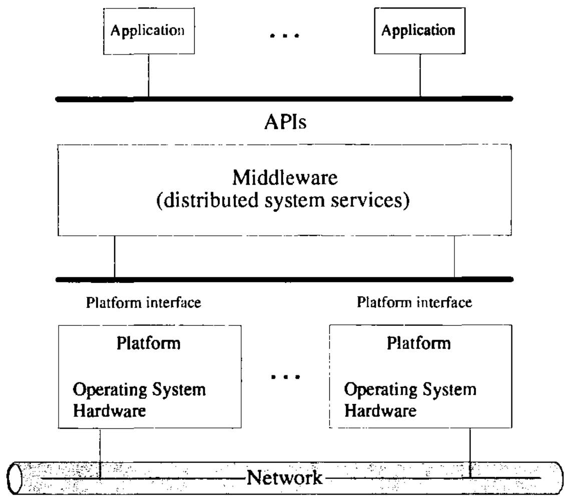


Figure 2.1: Middleware

Applications that communicate through a publisher-subscriber paradigm require the sending applications, or publishers, to publish messages without specifying recipients or having knowledge of the intended recipients. In a similar fashion, receiving applications, or subscribers, receive only those messages that the subscriber has registered for. This anonymous behavior provides a loosely coupled communication model between distributed applications.

In the publisher-subscriber communication scheme, applications can either be publishers, subscribers or both. The publisher and subscriber interaction is handled by the GridStat middleware, which acts as a forwarding agent to disseminate the events to the subscribers. The publishers and subscribers do not need to know each other, or need to actively participate in the interaction at the same time and neither entity is blocked while producing or consuming events.

The GridSt architecture, see figure 2.2, has two planes: a data plane and a management plane. The data plane consists of clouds of status routers and the management plane consists of a hierarchy of QoS brokers. The function of the data plane is to forward status events from the publishers to the subscribers. The function of the management plane is to manage resources in the data plane in order to provide QoS guarantees to admitted traffic flows.

Entities in GridStat include status variables, status events, publishers, subscribers, status routers edge status routers, and QoS brokers. A *status variable* represents some physical quantity (current, voltage, etc.) or state (breaker position open/closed, generator on/off, etc.) Each periodic measurement of the status variable is called a *status event.* Status events are placed into event messages, or GridStat packets, at a given rate and flow through the communication infrastructure from publishers to subscribers.

Publishers are the sources of status events and *subscribers* are the destinations of status events in the publisher-subscriber communication scheme. *Status routers* make up

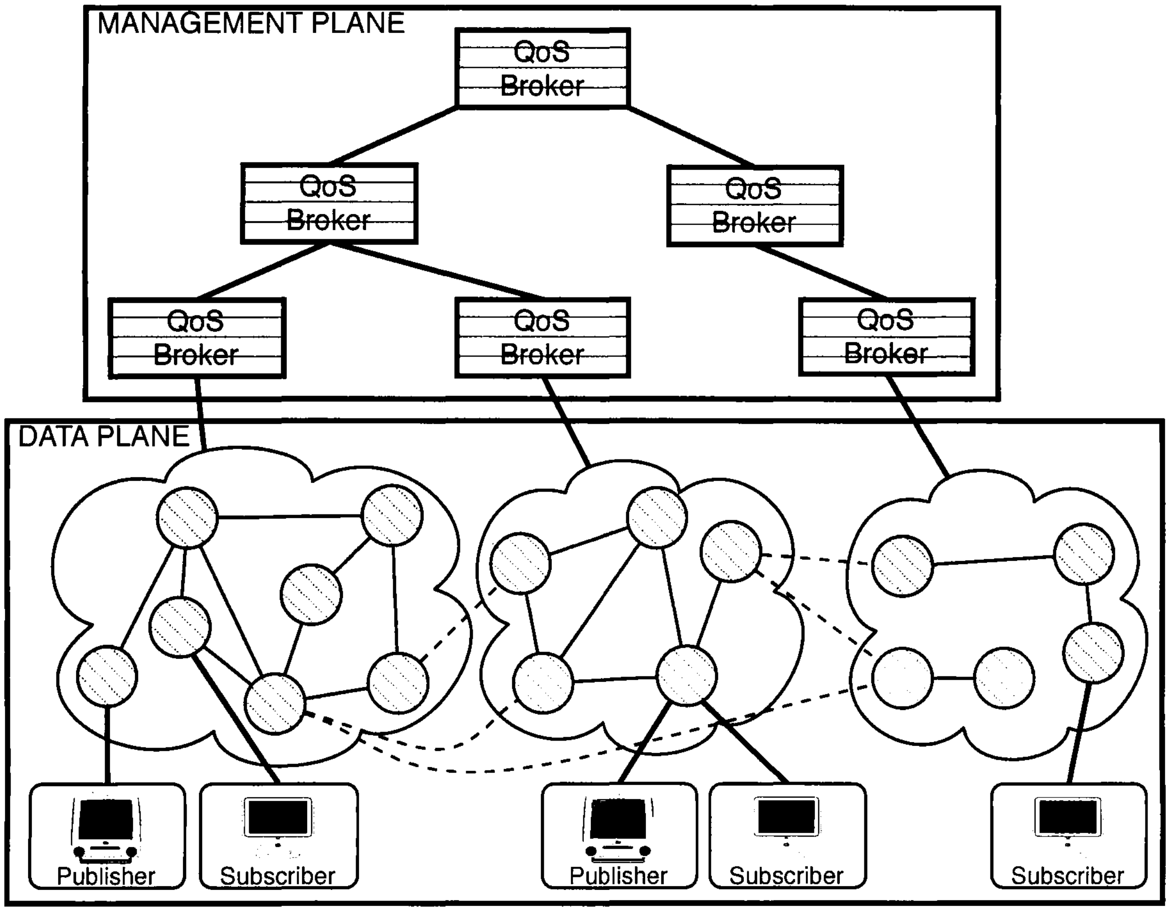


Figure 2.2: GridStat architecture, [Gje06]

the communication infrastructure and are connected to each other through point-to-point or overlay links. They serve as smart routers that store and forward status events, perform rate filtering (on the basis of timestamps contained in the packets), and support multicast routing. *Edge status routers* are just like status routers, but differ in that publishers and subscribers can only directly connect to edge status routers.

QoS brokers manage the operation of the data plane by making admission control, path establishment, and fault tolerance decisions. The lowest level QoS brokers are called leaf QoS brokers. Each leaf QoS broker manages one cloud, maintains the current state of the cloud network topology and available network resources.

2.5 Discussion

The aim of this research is to provide the appropriate level of QoS performance for the GridStat framework. It is clear that status dissemination for the electric power grid is a critical component for maintaining reliable power delivery and avoiding blackouts. It is consequently appropriate to classify GridStat as a hard real-time application due to the nature of what could happen in the event of failure to deliver status information by their time constraints.

An analysis by Hauser, et al., [HBD$^+$07], determines that the fundamental requirements for the performance of the control center are:

1. Control center displays for operators must accurately reflect the current system state so that all control decisions made by the operators are appropriate.
2. Substation equipment must carry out legitimate commands only, within specified time delays.

The implications of this analysis are that GridStat should be designed to support delay requirements ranging from a few tens of milliseconds for automated dynamic stabilization applications to a few seconds for conventional control center applications.

As a result, the approach presented in the remainder of this thesis is that GridStat should have a deterministic level of service due to the fact that it is best characterized as a hard real-time application. Fortunately, in GridStat admission of all flows allowed onto the network can be controlled and the scheduling algorithm used at the routers can be defined. Hence, we can determine if a flow will satisfy its Quality of Service delay requirement prior to network admission.

Chapter 3
Related work

The bulk of the related work has focused on providing QoS for the Internet. However these efforts have been less than successful mainly because the Internet was not originally designed to support QoS and is best-effort in nature.

3.1 Internet

The two highest profile efforts are IntServ and DiffServ resulting from standardization work within the Internet Engineering Task Force (IETF). The IETF divides the issue of providing QoS on the Internet into two areas: integrated and differentiated service.

3.1.1 Integrated Services

Integrated Services (IntServ), [BCS94], is a reservation based scheme intended to transform the Internet into an integrated-service communications infrastructure capable of supporting the transport of audio, video, real-time, and best-effort data traffic. The application frames its request within the Resource Reservation Protocol (RSVP), [Wro97b], and then passes this request to the network. Each router in the flow's path then performs an admission control test to determine if there are sufficient resources to meet the requested service without affecting the existing flows. The flow is only admitted if all the routers along the path have sufficient resources to meet the flow's requirements. That being the case, these resources are reserved for the life of the application or until renegotiated.

In addition to best-effort service, IntServ defines two other classes of service, Guaranteed Service and Predictive Service. The Guaranteed Service, [SPG97], provides a perfectly reliable upper bound on delay to support real-time applications. The Predictive Service, [Wro97a], supplies a fairly reliable, but not perfectly reliable, delay bound to applications that can tolerate some late packets.

The RSVP/Intserv approach suffers from some major problems identified by Y. Bernet et al., in RFC 2998, [BFY+00]:

1. The use of per-flow state and per-flow processing raises the possibility of scalability concerns for extremely large networks like the Internet.

2. Deployment in the routers must be widespread. For this scheme to be capable of producing the desired result from end-to-end, every router on each path to be used, must implement RSVP, admission control, classification and packet scheduling.

3. Deployment in applications must also be widespread. Each application that requires guaranteed service must make a reservation. Currently only a small number of applications have the ability to generate RSVP signaling. While that number could grow in the future, it is expected that many applications may never generate RSVP signaling.

Christin and Liebeherr, [CL02], contend that due to the unresolved issues regarding its scalability for large networks like the Internet, IntServ has not gained wide acceptance. In spite of these problems, at a high level the IntServ framework describes very closely what is needed to provide GridStat with real-time performance guarantees. The Intserv framework describes four components: the packet scheduler, the admission control routine, the classifier, and the resource reservation setup protocol.

The classifier idea from IntServ cannot be used in GridStat. At every IntServ router, each incoming packet gets mapped into a class. A class might correspond to a broad category of flows or a class might hold only a single flow. Then all packets in the same class get the same treatment from the packet scheduler. This is a priority based scheduling algorithm and cannot work to obtain delay guarantees for all packets in the Guaranteed Service level of service, as will be shown in section 3.4.1 of this chapter, if all packets in a class get the same treatment from the packet scheduler.

The component descriptive names that are capable of being transferred from the IntServ framework to GridStat are resource reservation, admission control, and packet scheduling. It should be noted that just the ideas of what needs to be done at a high level is transferred, not the actual IntServ components. To provide GridStat traffic flows with delay guarantees, the functions these three components provide will have to be implemented in the appropriate GridStat network elements.

3.1.2 Differentiated Services

Differentiated Services (DiffServ), [BBC+98], is a prioritization based approach to QoS on the Internet. DiffServ aims to provide QoS to a customer based on the customers aggregated traffic. The customer and provider negotiate a quantitative agreement (for example, 200 kbps, 10 kb burst) or qualitative agreement (for example, traffic treated as higher priority than normal traffic) for service and the provider provisions their network to offer such service. Each packet gets marked with a code to indicate the desired level of service.

Obviously, with this approach, the customer or application user is left with the responsibility of ensuring multiple applications originating from their network do not interfere because traffic characteristics are exceeding their specifications. DiffServ does not solve the QoS for Internet problem; it merely pushes the problem of managing time-sensitive traffic onto the user, where it still must be resolved.

3.2 Asynchronous Transfer Mode

Asynchronous Transfer Mode (ATM) is a connection-oriented approach to networking that uses a circuit switching design to guarantee QoS to constant bit rate and variable bit rate traffic flows. ATM attempts to combine the advantages of both circuit and packet switched techniques. Circuit switched networks offer guaranteed delivery and packet switched networks provide improved link utilization.

In ATM, a connection is known as a virtual circuit. Each virtual circuit is identified by a Virtual Circuit Identifier (VCI). A group of virtual circuits form a virtual path identified by a Virtual Path Identifier (VPI). Data is transmitted in fixed size cells that consist of a 48 byte payload and a 5 byte header. The header contains the VPI and VCI values of the connection on which the cell is sent.

When a request is made in ATM to establish a connection, a set of minimum acceptable QoS parameters is also specified. Typical parameters are cell rate, cell loss ratio, cell transfer delay, and cell delay variation. A connection is established only if the network can guarantee the requested QOS parameters.

Network Express (NetEx), [SLDZ97], is a communication software package, developed by the Real-Time Systems Group in the Department of Computer Science at Texas A&M University, that can provide connection-oriented delay guaranteed communication services at application level. With NetEx, user applications request a connection set-up by specifying their traffic, using some standard traffic descriptors, and Quality of Service. NetEx runs a connection admission control algorithm to check if the connection can be accepted. NetEx also has run-time traffic control to monitor for misbehaving traffic flows.

The only Quality of Service property that NetEx is setup to provide is end-to-end delay bound. In their paper, the only performance metric they test for is admission probability. They report findings on the relationship between admission probability and the number of connections generated per host. Therefore, it is impossible to measure precisely what NetEx can achieve in QoS performance.

While NetEx provides a service that is similiar to what is needed for GridStat, it is not an exact match for several reasons. Most importantly, NetEx does not support a multicast routing protocol. Also, Asynchronous Transfer Mode (ATM) is used in NetEx, whereas the current GridStat prototype uses User Datagram Protocol (UDP) packets over Internet Protocol (IP). And finally, NetEx has an internal route generation that finds one shortest path and each GridStat traffic flow each runs on two disjoint paths from source to destination with route found by shortest delay on the path.

3.3 Statistical Delay Guarantees

Considerable research has been done in the area of providing statistical delay guarantees. In this approach, the delay bounds are statistically guaranteed, in other words they are guaranteed with a reasonably high probability. This is expected to allow the links to operate at a higher utilization than if deterministically guaranteed QoS was used and still meet the QoS requirements for most of the traffic. However, the use of statistical delay guarantees has the obvious disadvantage of allowing a small fraction of the traffic to violate its QoS specifications. This is the reason GridStat does not use the statistical delay guarantees approach. GridStat is best characterized as a hard real-time application and cannot tolerate even a small fraction of its traffic to violate its QoS specifications.

Kurose, [Kur92], derived probabilistic bounds on delay and buffer occupancy of flows using the concept of stochastic ordering for network nodes that use FIFO scheduling. Reisslein et al., [RRR98, RRR99, RRR02], have derived statistical delay bounds for traffic flows in single node and multiple node settings. They approximate the loss probability at a link using independent Bernoulli random variables, with a fluid traffic model as opposed to a packetized model.

Elwalid and Mitra, [EM99], have developed a framework for Generalized Processor Sharing (GPS) scheduling which is based on statistical QoS guarantees and statistical multiplexing, using a fluid traffic model. The problem with GPS scheduling is that it is designed to allocate a fair share of the bandwidth to each flow source on a bit by bit basis and at the network level the smallest unit is a packet, not a bit.

Schemes for providing statistical QoS in networks using EDF scheduling were proposed by Sivaraman and Chiussi, [SC99], and Andrews, [And00]. And finally, Liebeherr et al., [LPY01], have proposed two network designs for statistical end-to-end delay guarantees, referred to as class-level aggregation and path-level aggregation.

3.4 Scheduling Based Approaches

The choice of a scheduling algorithm (or service discipline) is a critical decision when designing a real-time packet-switched network. The scheduling algorithm used in the router defines the order in which arriving packets gain access to the shared output link. Although packet scheduling is closely related to queuing theory, it is distinguished by the goal of minimizing worst case rather than average case delay within a real-time deterministic packet-switched network.

The First In First Out (FIFO), [DKS89], and Priority based, [Zha95], scheduling algorithms are simple and easy to implement, but in the presence of congestion at the routers unprotected flows can suffer starvation resulting in late packets or dropped packets due to finite buffer space. Many scheduling disciplines have been proposed recently that provide protection to well behaving flows and bound maximum delay, including Virtual Clock, [Zha90], Fair Queuing, [DKS89], along with Weighted Fair Queuing also called

Packetized Generalized Processor Sharing, [PG92, PG93], Worst-case Fair Weighted Fair Queuing, [BZ96], Self-Clocked Fair Queuing, [Gol94], Delay-Earliest Due Date, [FV90, KSF91, ZS94], Jitter-Earliest Due Date, [VZF91], Stop and Go, [Gol90], and Hierarchical Round Robin, [KKK90]. A comparison of several disciplines is given in, [ZK91, Zha95, ZK96].

To design new scheduling algorithms and to compare the existing ones with each other, it is helpful to define what are the desirable properties of a scheduling algorithm. In [Zha95], Zhang says for all scheduling disciplines, it is desirable that they are flexible and simple:

- *Flexibility*: The scheduling algorithm should be able to accommodate applications with varying traffic characteristics and performance requirements.
- *Simplicity*: The scheduling algorithm should be simple both conceptually and mechanically. Conceptual simplicity enables tractable analysis of the scheduling algorithm such that distributions or worst case bounds for performance parameters, such as delay, can be derived. Mechanical simplicity is desired to allow efficient implementation of the scheduling algorithm at high speed.

When Quality of Service performance guarantees in the network are needed, such as bounded delay, the following two scheduling algorithm properties are desirable:

- *Protection*: The scheduling algorithm should be able to protect the well behaving flows from different sources of variability, such as ill behaving flows and network load fluctuations. Ill behaving flows refer to flows that send more packets than their traffic profile allows. Network load fluctuations are caused by traffic bursts at a router. These bursts may happen even if the users meet their traffic constraints at the entrance of the network. Ideally the scheduling algorithm should be able to satisfy the performance requirements of all well behaving users even in the presence of these factors.

- *Efficiency*: A scheduling algorithm is more efficient than another one if it can meet the same end-to-end QoS performance guarantees under a heavier traffic load. To provide a meaningful comparison between scheduling algorithms, the concept of a schedulable region was formulated in [HLP91]. The schedulable region contains all possible combinations of flows that can be accepted onto the network without violating the end-to-end QoS performance guarantee of any other flow. A scheduling algorithm is more efficient than another if it has a larger schedulable region. Consequently, the use of a more efficient scheduling algorithm results in higher network utilization.

3.4.1 Basic Scheduling Algorithms

FIFO

The simplest scheduling algorithm is First In First Out (FIFO), see figure 3.1, sometimes called First Come First Served (FCFS). As packets arrive at a router, they are placed in an output queue and serviced from this queue based on their order of arrival. However, since the traffic flows are not classified in any way and the scheduling decisions

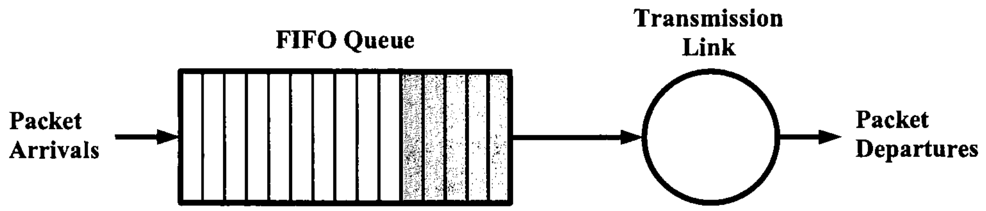


Figure 3.1: First In First Out (FIFO) scheduling

are based only on the arrival times, FIFO is not able to provide protection.

Priority based

In a priority scheduling algorithm, see figure 3.2, a higher priority traffic flow always has precedence over a lower priority flow. Upon arrival, packets are classified by priority depending on what flow they came from and placed into the appropriate queue. A priority scheduler will serve packets from the queue that is assigned with the highest priority. Queues with a lower priority are served only when all the queues with higher priority are empty. Among packets in the same priority class, the order in which they are served is typically FIFO.

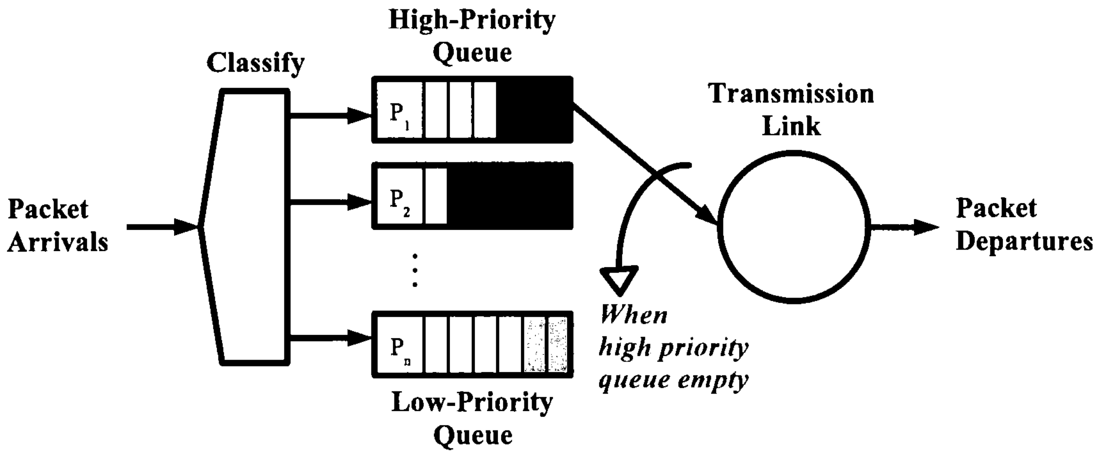


Figure 3.2: Priority scheduling

At a given priority level all flows are equivalent, just as in FIFO. Therefore, within priority levels and in general, priority scheduling is not able to provide protection.

Summary of FIFO and Priority Scheduling Algorithms

Both of the basic scheduling algorithms, FIFO and Priority, do not provide protection to existing flows in the presence of ill behaving flows and/or network load fluctuations. In the case of FIFO, a single misbehaving flow, sending packets at a sufficiently high speed, can capture an arbitrarily high fraction of the bandwidth of the outgoing link, [DKS89]. Priority scheduling suffers from the same lack of protection, due to FIFO scheduling being used within each priority.

If all well behaving flows in the network cannot be protected in the presence of ill behaving flows and/or network load fluctuations, then those flows cannot receive a guaranteed delay bound. With no protection, there can be no guarantee or bound on packet delay. Hence, the FIFO and Priority scheduling algorithms will not be further considered for use in providing per-connection end-to-end performance guarantees in a packet-switched network.

3.4.2 Work-Conserving or Non-Work-Conserving

Scheduling algorithms are classified as either work-conserving or non-work-conserving, depending on how they handle traffic distortions. Work-conserving scheduling algorithms accomodate traffic distortions while the non-work-conserving seek to control the distortions. With a work-conserving scheduling algorithm, a router is never idle when there is a packet to send.

In a router with a non-work-conserving (sometimes called rate-controlled) scheduling algorithm, the router may remain idle even if there are packets waiting to be sent. However, because these non-work-conserving scheduling algorithms may be forced to hold packets even when the output link is idle, they can increase the average end-to-end delay of all flows in the network and consequently have a lower utilization of network resources.

3.4.3 Work-Conserving Scheduling Algorithms

Virtual Clock

The Virtual Clock (VC) scheduling algorithm, [Zha90], is based on a priority queue. In this scheme, the ordering of packets in the priority queue is based on the time at which packets would have been sent if the router were using Time Division Multiplexing (TDM) to schedule packets. In TDM, time is divided into frames of fixed length. Each frame has a fixed number of constant-sized slots and each flow obtains one or more slots per frame.

VC protects flows from the effects of other flows, similar to TDM, but unlike TDM, the queue is work conserving since the TDM is only used to determine the order of service not when packets are allowed to be sent. VC takes advantage of any excess bandwidth which may be available whenever flows do not use their full time allotment.

The scheduling algorithm works by maintaining a real-time clock and two virtual timers for each flow, Virtual Clock (VC) and auxiliary Virtual Clock (auxVC), which are used to provide flow monitoring and packet scheduling. A flow is specified by its average rate (AR) and averaging interval (AI). Over each AI time period, dividing the total amount of data transmitted by AI should result in AR. Whenever a packet arrives the timers are advanced by the minimum packet spacing that maintains the average rate. Effectively this sets the timers to the earliest time that the next packet is eligible for sending. The packets are stamped with the auxVC time and placed in a sorted priority queue for sending.

Every AI $*$ AR packets, VC is compared to the real-time clock. If the VC is running ahead of the real-time clock, then the packets are arriving faster than agreed and action is taken to restrict the misbehaving flow. If it is running behind the real-time clock, then VC is set to the current real-time to prevent the flow from sending at a lower rate than AR in one interval followed by a higher rate than AR in the next without being detected.

The reason a separate virtual timer auxVC is needed is that a flow could send no packets for a long time and then send a large burst towards the end of AI and still meet its average rate when considered over the whole interval. Since the VC of the flow would not have advanced, it would be small compared to other flows that had been sending more regularly. The bursty flow would gain priority over the other flows since its packets would be sent first. It has effectively gained credit for not using its resources earlier. Since those resources can not be saved up, this scheduling algorithm must prevent such events. The auxVC timer prevents this by resetting itself to the real-time clock if this is greater than its current value at each packet arrival, thereby ensuring a bursty flow will not be serviced before other flows of equal or greater rate allocations.

When Virtual Clock was first proposed, no method was provided for determining the bounds on end-to-end delay. Since then, end-to-end delay bounds have been determined in [GLV97] and [FP95]. Extensions to VC have been proposed to allow for flows with variable rates in both non-work conserving, [LX95], and work conserving, [GV97], versions.

Fair Queuing and Weighted Fair Queuing

Fair Queuing (FQ), [DKS89], was introduced to allocate a fair share of the bandwidth to each flow source. FQ services each flow for a set amount of transmission time in a round robin fashion. Any unused transmission time is fairly distributed among all of the flows. One major problem is that FQ can not be implemented at the packet level unless all packets are the same length. Otherwise, a flow with long packets will again receive a disproportionate amount of the link bandwidth. Servicing packets on a bit by bit basis, which would be fair, is generally not practical.

FQ, as originally proposed, only provides a minimum guaranteed service. No bounds on delay or packet loss were derived. Bounds were derived for the rate-based source model by Parekh, [Par92], called Packet by Packet Generalized Processor Sharing (PGPS). Bit by bit fair queuing is also known as Generalized Processor Sharing (GPS).

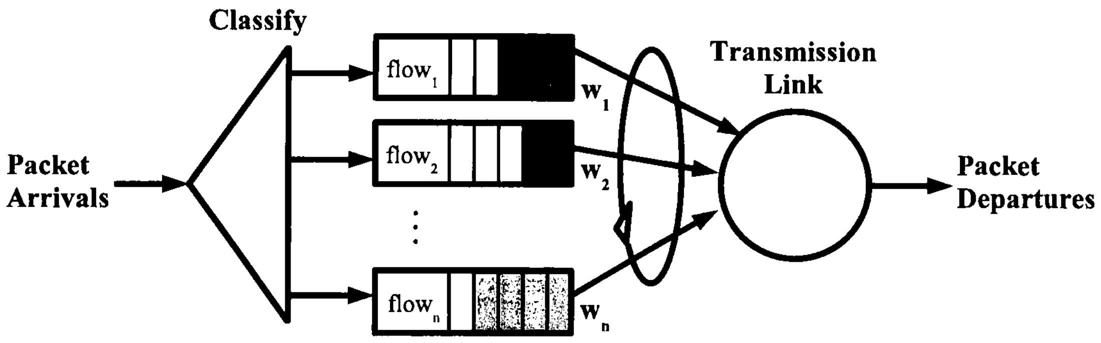


Figure 3.3: Weighted Fair Queueing scheduling

The approach in Weighted Fair Queueing (WFQ) scheduling, see figure 3.3, is to emulate the GPS system as much as possible. Arriving packets are classified and placed into their appropriate flow queue. In a round robin type fashion, the WFQ scheduler serves them – first flow 1, then flow 2, up to flow n. WFQ is work-conserving so it is never idle; the scheduler will immediately move to the next class in the sequence upon finding an empty flow queue.

In WFQ, each flow may receive a differential amount of service. Each flow i, is assigned a weight w_i. Even if all flows are queuing, flow i is still guaranteed to receive its share of the bandwidth.

[Kes97] provides an example. Let a leaky bucket constrained source i with parameters $(\sigma(i), \rho(i))$ pass through K schedulers, where the kth scheduler, $1 \leq k \leq K$, has link rate r(k). Where $\sigma(i)$ is the maximum burst size on flow i and $\rho(i)$ is the average data rate of flow i. Let g(i, k) be the service rate assigned to the connection at the kth scheduler, where:

$$g(i,k) = \phi(i,k)r(k)/\sum \phi(j,k)$$

Let g(i) be the smallest of the g(i, k)'s over all the schedulers. Assume that

$g(i) \geq \rho(i)$; otherwise the queue at one of the schedulers would increase without bound. If the largest packet allowed on the flow is of size Pmax(i) and the largest packet allowed on the network is Pmax, then independent of the number of schedulers traversed and independent of the behavior of the other flows sharing the path, the worst-case end-to-end queuing and transmission delay D_i is bounded by:

$$D_i \leq \frac{\sigma(i)}{g(i)} + \sum_{k=1}^{K-1} \frac{Pmax_i}{g(i,k)} + \sum_{k=1}^{K} \frac{Pmax}{r(k)}$$

Worst-case Fair Weighted Fair Queuing

Worst-case Fair Weighted Fair Queuing (WF^2Q), [BZ96], is the same as WFQ, except that the scheduler chooses the packet with the smallest finish time among all the packets that would have already started service in the corresponding GPS emulation. It was shown by Zhang, [Zha95], that the service order of packets under WFQ and WF^2Q system can be different for the same traffic arrival pattern.

At any given time, the accumulated service provided for each flow by either WF^2Q or WFQ never falls behind GPS by more than one packet size. It can be shown, [Zha95], that the difference between services provided by WF^2Q and GPS is always less than one packet size. Also, in the worst case both WF^2Q and WFQ can fall behind GPS by the same amount, so they provide the same end-to-end delay bounds.

Self-Clocked Fair Queuing

Self-Clocked Fair Queuing (SCFQ), [Gol94], is the same as WFQ, except virtual time computation. The scheme in SCFQ is to simplify the computation by estimating the system virtual time V(t) with the virtual service time of the packet that is currently being served in GPS.

The SCFQ algorithm has the following steps:

1. Each arriving packet p_k^i is tagged with a service tag F_k^i before it is placed in the queue. The packets in the queue are then picked up for service in increasing order of the associated service tags.

2. For each flow k, the service tags of the arriving packets are computed as:

$$F_k^i = \frac{1}{r_k} L_k^i + \max\left(F_k^{i-1}, v(a_k^i)\right)$$

   where $F_k^0 = 0$.

3. v(t), the system's virtual time at time t, is defined as being equal to the service tag of the packet receiving service at that time.

4. Once a busy period is over and the router is free (no more packets in the queue), the algorithm is reinitialized by setting v(t) to zero and the packet counts i to zero for each flow k .

Delay-Earliest Due Date

Delay-Earliest Due Date (Delay-EDD), [FV90], is an extension of the Earliest Due Date (EDD) scheduling algorithm. In EDD, each packet is assigned a deadline and the scheduler servers packets in order of their deadlines.

In Delay-EDD, see figure 3.4, each traffic flow must negotiate a service contract with each scheduler on its path from source to destination. The contract states that if a source obeys some peak rate, then every packet on that flow receives a worst case delay smaller than some bound. For admission to the network two conditions must be satisfied, the sum of the peak rates is smaller than the link capacity for all links on the path and that even in the worst case, with all flows sending at their peak rates, the delay bound is still met at each router (schedulability test).

Delay-EDD scheduling algorithm assigns deadlines to packets in the following manner. It sets a packet's deadline to the time at which it should have been sent had it been received according to the flow's contract, which is slower than its peak rate. By reserving bandwidth at the connection's peak rate, a Delay-EDD scheduler can ensure it has served the previous packet from that flow before the next packet arrives. So, every packet from a flow obeying its peak rate constraint receives a hard delay bound.

The Delay-EDD expected deadline state variable for flow i, ExD_i^k, is stored at router k and is calculated as follows:

$$\mathrm{ExD}_i^k = \max(\mathrm{AT}_i^k + \mathrm{d}_i^k, \mathrm{ExD}_i^k + \mathrm{Xmin}_i)$$

Where AT_i^k is the arrival time of the packet and Xmin_i is the minimum packet inter arrival time and d_i^k is the local delay bound for flow i at router k.

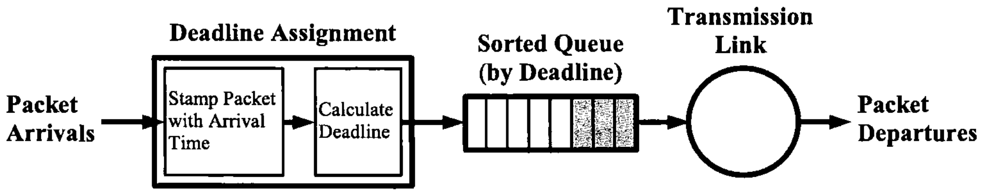


Figure 3.4: Delay-EDD scheduling

Since the assignment of deadlines in delay-EDD is based on the two parameters, Xmin_i and d_i^k, delay requirements are decoupled from bandwidth requirements. For example, a flow reserving a small bandwidth could obtain a small delay bound. In summary, Delay EDD decouples the delay and bandwidth bounds, but at the cost of reserving bandwidth at the peak rate.

3.4.4 Non-Work-Conserving Scheduling Algorithms

Jitter-Earliest Due Date

The Jitter-EDD scheduling algorithm, [VZF91], extends Delay-EDD to provide delay jitter bounds (a bound on both the minimum and the maximum delay). In Jitter-EDD, a delay jitter regulator is installed before the EDD scheduler. With the regulator, all packets receive the same delay at each hop (except for the last hop), so the delay jitter along the path from source to destination is reduced to the delay jitter on the last hop. The Jitter-EDD approach reduces the overall buffering required at each router by reducing the number of packets which could potentially arrive early (reduces the worst case scenario).

Stop and Go

The Stop and Go, [Gol90], scheduling algorithm bounds flow delays by dividing a links sending time into fixed size frames and assigning flows to frames whose sending rate matches the delay bound requirements of the flow. In each frame, only packets that arrive at the router in the previous frame are sent out. Within frames, the service order of packets is arbitrary. It can be shown, with this scheduling algorithm, that packets on a flow receive both a minimum and a maximum delay as they go from source to destination.

Hierarchical Round Robin

In Hierarchical Round Robin (HRR), [KKK90], there exists a hierarchy of service levels, with each level having a fixed number of slots. The levels are numbered 1 ... N, with the highest rate flows at level 1. A flow is allocated a given number of slots at a selected level. The scheduler cycles through the slots at each level, see figure 3.5, in a round robin fashion. The time to service all the slots at a given level is called the Frame Time (FT) at that level. The total link bandwidth is partitioned in among these levels.

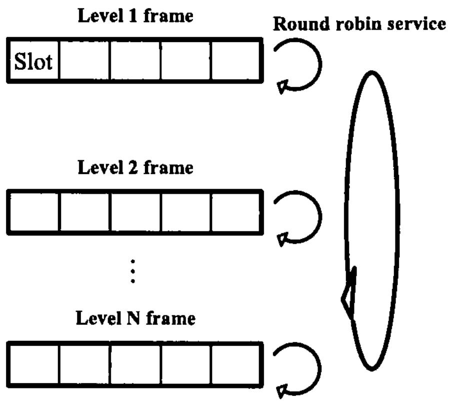


Figure 3.5: Hierarchical Round Robin (HRR) scheduling, [ZK91]

The HRR scheduling algorithm can provide guaranteed bandwidth to rate controlled flows. The bandwidth received by flow j at level i is $\frac{a_j}{FT_i}$ slots/sec. Where a_j is the service quantum for flow j. Since HRR always completes one round through its slots once every frame time, it can provide a maximum delay bound to the flows allocated to that level.

3.4.5 End-to-End Considerations

One method for determining end-to-end delay bounds is to consider each server in isolation, and then compute the summation of the maximum delay at each server along the path from the source to the destination, [Cru91a, Cru91b]. However, this method has the following disadvantages:

- Due to varying queue sizes, the delay experienced by packets at a router will vary. Consequently, the shape of the traffic flow can become distorted as it travels through the network. Therefore, even if a source traffic specification is known at the first router on the path of the flow, it is difficult to determine the exact specification at a router further down on the path. This makes determining the delay at each of the routers on the path difficult.

- In many scheduling algorithms, if a packet experiences a high delay at a router, it may experience a lower delay at the next router along the path. If each router is considered by itself, the dependence between the delay experienced by packets at different routers is not accounted for, and the bound on the delay at that router will probably be very conservative.

Goyal, et al., [GLV97], generalize the previous approach to a heterogeneous network of servers each of which uses a scheduling algorithm in GR for any source specification. They derive a delay guarantee for a network of routers and reduce the problem of determining the end-to-end delay to that of determining delay at a single router.

To determine an upper bound on the end-to-end delay of packets of a flow, [GLV97], consider a flow which is served by K servers. Let server 0 be the source and server $K+1$ be the destination. Let d^j be the delay experienced by the j^{th} packet of a flow. Since server K guarantees that packet p^j will be transmitted by time $GRC^K(p^j, r^{j,K}) + \beta^K$ and the packet arrives at the first node at time $A^1(p^j)$, the result is:

$$d^j \leq GRC^K(p^j, r^{j,K}) + \alpha^K - A^1(p^j)$$

where $\alpha^K = \beta^K + \tau^{K,K+1}$. Note that $GRC^K(p^j, \hat{r}^j) - GRC^K(p^j, r^{j,K}) \geq \frac{l^j}{\hat{r}^j} - \frac{l^j}{r^{j,K}}$.

Therefore,

$$d^j \leq GRC^K(p^j, \hat{r}^{j,K}) - \left(\frac{l^j}{\hat{r}^j} - \frac{l^j}{r^{j,K}}\right) + \alpha^K - A^1(p^j)$$

If each router on the path of the flow uses a scheduling algorithm in GR, then given the path configuration, $GRC^K(p^j, \hat{r}^j)$ can be related to $GRC^1(p^j, \hat{r}^j)$. When packet fragmentation and reassembly does not occur, then using results derived by Goyal, et al.:

$$d^j \leq \left(GRC^1(p^j, \hat{r}^j) - A^1(p^j)\right) + \left(\sum_{i=1}^{K-1} \max_{n\in[1..j]} \frac{l^n}{r^{n,i}} - \left(\frac{l^j}{\hat{r}^j} - \frac{l^j}{r^{j,K}}\right)\right) + \left(\sum_{i=1}^{K} \alpha^n\right)$$

Therefore, the end-to-end delay of a packet consists of the following three components:

1. $\sum_{n=1}^{n=K} \alpha^n$: Since $\alpha^n = \beta^n + \tau^{n,n+1}$, this term is known because it is completely characterized by the scheduling algorithm used in the routers and the propagation delay in the network.

2. $\sum_{i=1}^{K-1} \max_{n\in[1..j]} \frac{l^n}{r^{n,i}} - \left(\frac{l^j}{\hat{r}^j} - \frac{l^j}{r^{j,K}}\right)$: This term depends on the length of the packets and the rate allocated to it at the routers. Therefore, this term is known if the length of the packets transmitted and the rate assignments are both known.

3. $GRC^1(p^j, \hat{r}^j) - A^1(p^j)$: This term depends on arrival process characteristics of a flow.

Only the third term depends on arrival process characteristics of a flow. If fixed rate is assigned to all the packets of a flow, then this term is the queuing delay at server with capacity $\hat{r}$ where $\hat{r}$ is the bottleneck rate for the flow. Therefore, the network can be abstracted as a single server, with either fixed or variable capacity.

Determining end-to-end delay is reduced to determining the delay at a single router. If a flow is characterized using a deterministic traffic specification, then single server queuing analysis can be used to determine an upper bound on the delay of packets of a flow.

Chapter 4
Conceptional Structure for Delay Guarantees

Any framework that provides delay guarantees needs an admission control policy or mechanism on the control path and an appropriate packet scheduling algorithm on the data path. The admission control admits a flow only if enough resources are available in the network, while the scheduling algorithm assures that all guarantees will be met for well behaved flows as their packets travel from source to destination. In GridStat, the QoS brokers manage the admission control process and related sub-tasks.

4.1 Selection of a Scheduling Algorithm for GridStat

GridStat has a number of unique characteristics that make it easier to provide real-time guarantees:

- Static route packet switching is used, as opposed to dynamic routing.
- Jitter does not need to be bounded. The only concern in this case is guaranteeing that an end-to-end delay bound is never exceeded.
- It presumes the use of a dedicated network for real-time traffic, so traffic allowed onto the network is tightly controlled.
- Rate-filtering is performed in the routers, so the possibility of ill behaving flows on the network is greatly reduced.

First of all, the scheduling algorithm should belong to the Guaranteed Rate (GR) class, [GV97], in order for all packets to obtain delay guarantees when used in conjunction with the necessary elements of a dedicated network, the use of admission control, and resource reservation. In addition, work-conserving scheduling algorithms are preferred over non-work-conserving scheduling algorithms for GridStat because they deliver better average case end-to-end delay bounds.

Work-conserving scheduling algorithms that belong to GR, include Virtual Clock (VC), WFQ, WF^{2}Q, Self-Clocked Fair Queuing (SCFQ), and Delay-EDD. Although it would be possible to use any of these five scheduling algorithms for GridStat, it is desirable that the one selected be efficient, protective, flexible, and simple (mechanically and conceptually) as described in the previous chapter.

Efficient: For VC, WFQ, WF^{2}Q, and SCFQ the equation to update the priority index has only one rate parameter. Having only one rate parameter introduces the problem of coupling between the allocation of delay bound and bandwidth.

For example, when VC, WFQ and WF^{2}Q are leaky-bucket constrainted with burst size σ_j and rate ρ_j for flow j, the end-to-end delay bound is $\frac{\sigma_j + n \cdot L_{max}}{r_j} + \sum_{i=1}^{n} \frac{L_{max}}{C_i}$, where n is the number of routers traversed by the flow, r_j is the guaranteed rate for the flow, C_i is the link speed of the ith router and L_{max} is the largest packet size. Notice that the end-to-end delay bound is inversely proportional to the guaranteed rate. Therefore, in order for a flow to get a low delay bound, high bandwidth needs to be allocated.

It has been shown that a strategy that decouples delay and rate allocation, such as in Delay-EDD, can result in higher utilization of the network, [Zha95]. Therefore, Delay-EDD is rated as efficient and the other four are rated as not efficient.

Protective: All five scheduling algorithms provide protection to admitted flows. For them to provide guaranteed performance, resources are reserved for a flow prior to admission and a scheduling algorithm belonging to GR is used. Based on this, all five scheduling algorithms can guarantee a deadline, or delay guarantee, by which a packet of a flow will be transmitted and consequently protection as well.

Flexible: Due to the specialized nature of GridStat, all five scheduling algorithms are rated the same. This is because all traffic flows on the data plane need bounded end-to-end delay guarantees. Different levels of QoS do not need to be supported as other applications may require.

Mechanically Simple: All five of the scheduling algorithms use a sorted priority queue. The insertion operation being O(logN), where N is the number of packets in the queue. This means that none of the scheduling algorithms can be considered mechanically simple. However, some are more simple than others.

For VC and Delay-EDD that use real time to compute their priority index, the computation is relatively straightforward and they are rated moderately mechanically simple. For WFQ and WF^{2}Q, which use virtual times from an emulated reference system, the computation is more complex and for that reason they are rated the least mechanically simple. For SCFQ, the virtual time is defined as being equal to the service tag of the packet receiving service at that time. Although SCFQ uses virtual time, it has a straightforward determination of virtual time and is therefore rated as being moderate in terms of mechanical simplicity.

Conceptually Simple: Conceptually, there is not one that stands out as being very simple or very complex relative to the others. So, all are rated as moderate in this property.

In the final analysis, WFQ and WF^{2}Q can be ruled out as being too mechanically complex relative to the other scheduling algorithms. For flexibility and conceptual simplicity, there is no significant difference between the scheduling algorithms. With the same guaranteed rate, the delay bound provided by SCFQ is larger than the one provided by VC, [Zha95]. Therefore, SCFQ can be ruled out as a possibility.

The two remaining choices are VC and Delay-EDD. With VC and Delay-EDD roughly the same in terms of mechanical simplicity, it then is left to efficiency to determine which is the better one to use. Delay-EDD is better in terms of efficiency due to the decoupling of delay and rate allocation. Hence, Delay-EDD is best suited to be used in the GridStat routers, because the use of Delay-EDD will result in a higher utilization of the network.

4.2 Admission Control

Admission control is required to regulate the number of traffic flows in the system so the delay guarantees of all properly admitted flows will be satisfied. The procedure, as shown in figure 4.1, for admission of a new traffic flow is based on the admission control module proposed by Gjermundrød et al., [GDB+03]. The admission control procedure in GridStat is administered by the QoS brokers and has four main steps: Request for Subscription, Path Selection, Path Evaluation, and Path Establishment.

4.2.1 Request for Subscription

In the GridStat model, the subscription interval is decoupled from the publication interval. However, when requesting a subscription, the subscriber needs to declare the desired interval so the edge status router doing the rate filtering for that publisher will perform correctly.

In the request for subscription, the Subscriber asks for subscription to some Publisher and provides the following information:

1. Traffic flow characteristics:
   (a) X_{min} – minimal packet interarrival time
   (b) X_{ave} – average packet interarrival time
   (c) I – interval over which X_{ave} is computed
2. Requested maximum end-to-end delay (D).

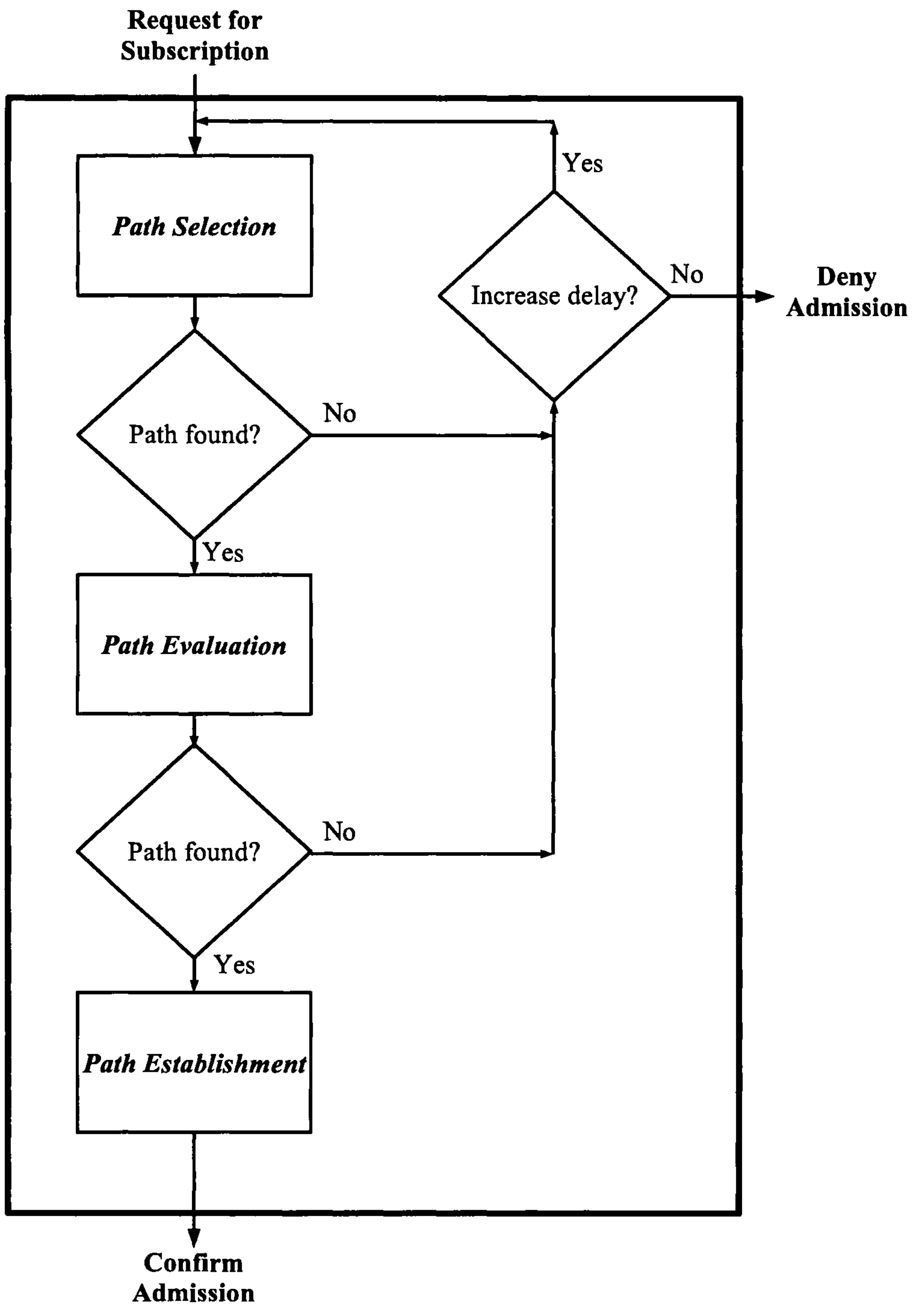


Figure 4.1: Admission control flowchart

4.2.2 Path Selection

To convey status data in a reliable manner and to provide a measure of fault tolerance, GridStat uses two disjoint paths from source to destination. Therefore, Path Selection deals with finding two disjoint paths in the network.

The problem of finding a delay-constrained minimum cost path is NP-complete, so the common approach is to use a heuristic in order to keep the computational cost low. A method for finding at least two paths from the source to each destination in the network, called Dynamic Weight Heuristic (DWH), was proposed by Irava, [Ira06], in his PhD thesis. DWH produces low cost, delay-constrained multicast trees and was specifically designed to be used in GridStat. Evaluations contained in his thesis show that DWH can construct multicast graphs with 10-15% lower costs than the node-priority based heuristics used in QDMR.

4.2.3 Path Evaluation

Before the two disjoint paths can be established, there needs to be a determination if there exists sufficient resources available at each router on the paths, for the new flow. A message must be sent to the QoS Brokers assigned to the routers along the proposed paths, checking that each router has sufficient available resources.

This is a two round process for each disjoint path. On the first round, the new flow is given the lowest available local delay bound at each router as returned by schedulability analysis, see section 4.3. Then sufficient bandwidth and buffer space for the new flow are tentatively reserved. At any router on the path, if the sum of local delays exceeds the requested maximum end-to-end delay, the path evaluation will fail and there will be an opportunity for the requested maximum end-to-end delay to be increased and the path evaluation will restart with the new request. If the requested maximum end-to-end delay is not increased, the admission for this new flow will be denied.

The second round is used to apportion the delay slack, if there is any. Slack is defined by the following equation:

$$\mathrm{Slack} = \mathrm{D} - \sum_{i=1}^{n} \mathrm{d_i}$$

where D is requested end-to-end delay, n is the total number of routers on the path, and d_i is the tentatively reserved local delay bound at the i^{th} router.

The responsibility of the second round is to distribute the slack among all the routers on the path. Apportioning of the slack could be done in many different ways, for example: straight percentage division to each router, percentage at each router could be determined by how loaded the router was relative to the other routers on the path, or the slack could be distributed in a linear fashion (with a smaller percentage close to the source and becoming larger approaching the destination).

To keep the implementation simple, a straight percentage division is used. Just divide total slack by the number of routers on the path.

$$\mathrm{slack_i} = \frac{\mathrm{Slack}}{\mathrm{n}}$$

where Slack is the total slack on the path and n is the number of routers on the path. Therefore $\mathrm{slack_i}$ is the amount of Slack to be distributed to the $\mathrm{i^{th}}$ router.

4.2.4 Path Establishment

Now that sufficient resources are determined to be available at each router along the entire path, commit the reserved resources at each router to the new traffic flow. By ensuring the local delay requirements are met at each router, the end-to-end delay bounds are guaranteed to be satisfied. At the conclusion of this step, all routers on the path are notified so they can adjust their rate filtering parameter for this flow.

4.3 Schedulability Analysis for Delay-EDD

Schedulability analysis determines whether the performance which can be offered at a particular router is sufficient to meet all delay requirements based on the traffic flow characteristics of the incoming traffic, both existing and new.

A set of n channels $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, \ldots, n$, is said to be ordered if $d_1 \leq d_2 \leq \cdots \leq d_n$.

The following theorem, [ZS94], provides a means of checking the schedulability of flows through a router under a non-preemptive deadline scheduling policy, such as Delay-EDD. It answers the problem: Suppose a set of $n - 1$ ordered channels, $\tau_i = (T_i, C_i, d_i), i = 1, 2, \ldots, n - 1$, are strongly schedulable over a link. Given a new channel τ_n with minimum packet interarrival time T_n and maximum packet transmission time C_n, what is the minimum value of d_n, such that all $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, \ldots, n$ are still strongly schedulable?

Theorem: Let $d_n = C_n + C_p$. If for $i = 0, \ldots, n$, $k = 1, \ldots, n$,

$$\delta_k(i) = \sum_{d_j \leq \min\{d_i, d_k\}} (1 + (d_k - d_j)/T_j)C_j + C_i - d_k \leq 0,$$

then $d_n = C_n + C_p$ is the solution to the problem. Otherwise, let $K_G^i = \{k : \delta_k(i) > 0$. The solution to the problem is:

$$d_n = \max_{0 \leq i \leq n} \{\max\{d_n^k(i) : k \in K_G^i\}\},$$

where $d_n^k(i) = C_n + (T_n/C_n)\delta_k(i)$ if $C_n + (T_n/C_n)\delta_k(i) < d_k$, otherwise,

$d_n^k = d_k + (C_n - d_k + (T_n/C_n)\delta_k(i))/(1 + (1 - \sum_{d_i \le d_k} C_i/T_i)(T_n/C_n))$.

4.4 Delay-EDD Scheduling in the Status Routers

The details of how Delay-EDD Scheduling in the status routers was implemented is presented in the next chapter. The key parts include time-stamping the packet upon arrival at the router, calculating the local deadline for that packet by updating the flows state variable for its expected deadline, and placing the packet into a sorted queue at the output link, such that the packet with the earliest deadline due gets sent first.

Chapter 5
Implementation

This chapter presents the implementation of the Delay EDD scheduling algorithm in the status routers of the GridStat framework. The first and second sections describe in more detail exactly how the packet scheduling fits into GridStat. The third section describes how the status router Java code was changed from FIFO to Delay EDD.

5.1 GridStat architecture

GridStat uses the User Datagram Protocol (RFC 768) and Internet Protocol (RFC 791) to communicate between GridStat data plane entities such as publishers, subscribers, and routers. A GridStat packet, see table 4.1, has the following elements:

Bytes	Description
8	timestamp
4	pubID
1	number of optional fields
3	padding
8	user data
0 to 96	0 to 4 24-byte optional user data fields

Table 5.1: GridStat packet

Internally, in the current Java implementation of GridStat, the class DatagramChannel (java.nio.channels.DatagramChannel) is used for input and output of UDP packets. The class DatagramChannel extends AbstractSelectableChannel and implements ByteChannel, ScatteringByteChannel, GatheringByteChannel. First the DatagramSocket class is used to bind a channel to a port. Then to use the

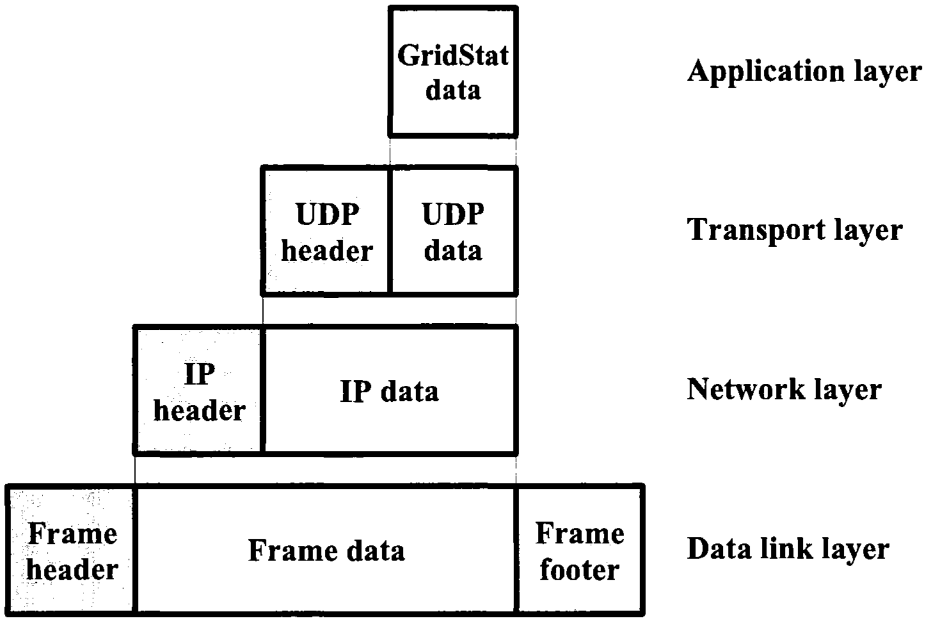


Figure 5.1: Encapsulation into UDP datagram

DatagramChannel the program reads and writes ByteBuffers, in the same way as is done with a SocketChannel.

Figure 5.1 shows the encapsulation of a GridStat packet into a UDP datagram. On the wire the total packet size (with zero added optional user data fields) is 78 bytes, with IP header and Ethernet preamble, header and footer. This includes 24 bytes for the GridStat data, 8 bytes for UDP header, 20 bytes for IP header, and 26 bytes for the Ethernet preamble, header and footer.

5.2 FIFO Status Router

The data flow through the FIFO status router, see figure 5.2, is as follows. The EventChannelSR gets a buffer from the BufferCache and uses the read() method of DatagramChannel to read in a packet from the packet arrivals. The packet is then sent to *the Routing*Tbl using routeEvent(), which places a reference to the buffer into each

SendingThread that the packet is to be routed to. The packet is then stored by SendingThread in a ring buffer (FIFO queue) until its turn.

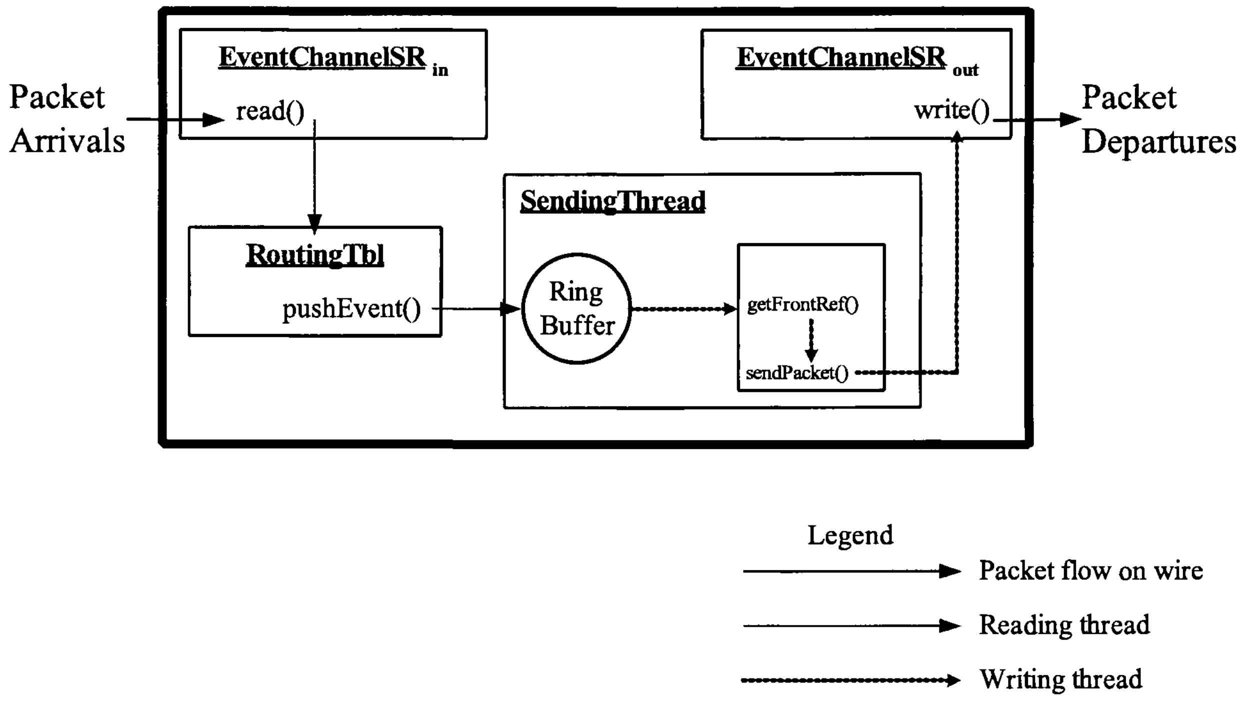


Figure 5.2: FIFO Status Router

In SendingThread.eventThreadDirect() a new Thread is created and started to get packets from the ring buffer and send them on the outgoing channel. This Thread uses getFrontRef() to get a reference for the packet from the ring buffer and forwards the packet, using sendPacket(), to the EventChannelSR which uses the write() method of DatagramChannel to write the packet to the appropriate outgoing channel. Then the packet is removed and recycled from the buffer by removeFront().

5.3 Delay EDD Status Router

Four key modifications are needed to change the status router from FIFO queuing to Delay EDD queueing, see figure 5.3. The incoming packet gets time stamped upon arrival at the router. The state variable that holds the expected deadline for this flow is updated according to the Delay EDD formula as described in Chapter 3. Then packets are added to the queue in sorted order by their updated expected deadline, ExD. With packets being sent on the outgoing link by order of earliest deadline in the queue goes first. The packing of multiple events into one packet was eliminated due to the work conserving nature of Delay EDD.

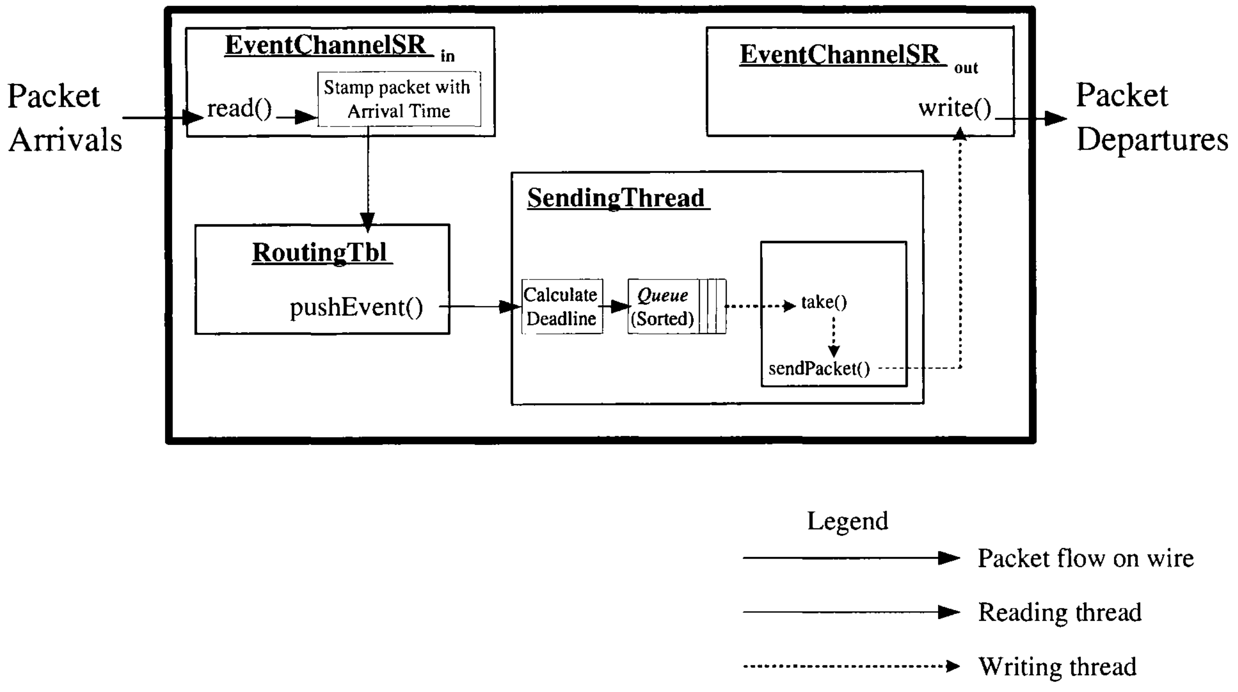


Figure 5.3: Delay EDD Status Router

5.3.1 Deadline assignment

At each router there is a state variable for the Expected Deadline (ExD) associated with each flow. A hash table data structure was used to store the state variables, with the packet flow identifier being used as the key into the hash table. The advantage of using a hash table is that it will provide constant-time, or O(1), lookup on average.

The Java collection class ConcurrentHashMap was used to store the state variable, ExD, necessary for the Delay-EDD scheduling algorithm. The class ExpectedDeadline was created for the state variable ExD, see figure 5.4. ConcurrentHashMap is thread-safe and provides its own synchronization for concurrent access or updates.

When a packet arrives at the status router, it is time stamped with Arrival Time. The Arrival Time used is the time returned by the Java System call nanoTime(). Then its Expected Deadline is looked up in the hash table using the packet flow identifier as the key into the table. Then using the packet Arrival Time, the ExD is updated. The updated ExD is stored into the hash table and attached to the packet. Note that the deadline is not stored in the packet, it just goes along with the packet until the packet is sent on the appropriate outgoing physical link.

5.3.2 Sorted Queue

The Java collection class PriorityBlockingQueue was used for the class for the Delay EDD queue. This is a thread safe version of the Java collection class PriorityQueue. The implementation for PriorityQueue provides O(log(n)) time for the insertion method add() and constant time for the take() method.

To specify correct ordering when using the SendHolder objects in the collection, it was necessary for SendHolder to implement Comparable and override the compareTo(), equals(), and hashCode() methods, see figure 5.5. In a PriorityBlockingQueue the head is

```
public final class ExpectedDeadline
{
      /**
       * The <code>ExpectedDeadline</code> default constructor for this class.
       */
      public ExpectedDeadline()
      {
            this.m_variableId = 0;
            this.m_ExD = 0;
            this.m_delay = 0;
            this.m_Xmin = 0;
      }

      /**
       * The <code>ExpectedDeadline</code> constructor for this class.
       */
      public ExpectedDeadline( long variableId, long delay, long Xmin )
      {
            this.m_variableId = variableId;
            this.m_ExD = 0;
            this.m_delay = delay;
            this.m_Xmin = Xmin;
      }

      /**
       * The <code>updateDeadline</code> method is used to update this state variable.
       * <BR>
       * @param arrivalTime The arrival time for the packet at this status router.
       * @return Returns m_ExD, the updated expected deadline.
       */
      public long updateDeadline( long arrivalTime )
      {
            if ( ( arrivalTime + m_delay ) > ( m_ExD + m_Xmin ) )
            {
                  m_ExD = arrivalTime + m_delay;
            }
            else
            {
                  m_ExD = m_ExD + m_Xmin;
            }

            return m_ExD;
      }

      /**
       * The <code>getVariableId</code> method is used to get the variableId.
       * <BR>
       * @return Returns m_variableId.
       */
      public long getVariableId()
      {
            return m_variableId;
      }

      /**
       * The <code>getExpectedDeadline</code> method is used to get the ExpectedDeadline.
       * <BR>
       * @return Returns m_ExD.
       */
      public long getExpectedDeadline()
      {
            return m_ExD;
      }


      ///////////////////////////////////////////////////////////////////////////////
      // Attributes
      private long m_variableId;
      private long m_ExD;
      private long m_delay;
      private long m_Xmin;
}
```

Figure 5.4: ExpectedDeadline class

the least element with respect to the specified ordering.

```
public int compareTo(Object o)
{
        if (!(o instanceof SendHolder))
        {
                throw new ClassCastException();
        }
        if (((SendHolder)o).m_deadline < this.m_deadline)
        {
                return 1;
        }
        if (((SendHolder)o).m_deadline > this.m_deadline)
        {
                return -1;
        }

        return 0;
}
public boolean equals(Object o)
{
        if (!(o instanceof SendHolder))
        {
                return false;
        }
        if (((SendHolder)o).m_deadline == this.m_deadline)
        {
                return true;
        }

        return false;
}
public int hashCode()
{
        return (int)(this.m_deadline^(this.m_deadline>>>32));
}
```

Figure 5.5: Override SendHolder methods for correct ordering

In this way the add() and take() methods of PriorityBlockingQueue to work correctly. When a SendHolder object is added to the queue, it is inserted into the sorted queue ordered by the expected deadline, ExD. And when take() is invoked, it retrieves and removes the head of the queue, waiting if no elements are present on the queue.

5.3.3 Packing of events

Delay EDD is a work conserving discipline and as such packing of multiple events into one packet at the status router is contrary to that objective. It only remains work conserving if an output link is kept busy as long as there are packets addressed to the output, in the sorted queue. Each packet needs to be treated individually as it arrives at the status router, gets stamped with a deadline, and placed into the sorted queue for its outgoing link. Therefore, in the constants.java file,

MAX_EVENT_ELEMENT_PER_PACKET is set equal to 1.

Chapter 6
Experimental Evaluation

In this chapter experimental results are obtained on the performance of the GridStat prototype, as currently implemented in Java. The main purpose of the experiments is to compare the end-to-end delay characteristics of the two scheduling algorithms, Delay-Earliest Due Date (Delay EDD) and First In First Out (FIFO). A secondary and related purpose is to compare the local delay of the two scheduling algorithms, within an individual status router.

6.1 Experimental Setup

The testbed used for these experiments is located in the Engineering Research/Teaching Laboratory (ETRL) network infrastructure lab, or "niflab", at Washingtion State University. The experiments were conducted on five identical HP Vectra VE computers, with the following specifications:

- 650 MHz Pentium III processor
- 512 MB SDRAM
- 8.4 GB Hard Drive
- Linux operating system, kernel 2.6.9
- Java Virtual Machine jdk1.5.0_06, with Java HotSpot Server VM enabled

The Linux nodes are all connected via a HP Procurve 2424M switch, providing 100 Mbps bandwidth between nodes.

6.2 Experiments and results

The experimental topology for the first four experiments is shown in Figure 6.1.

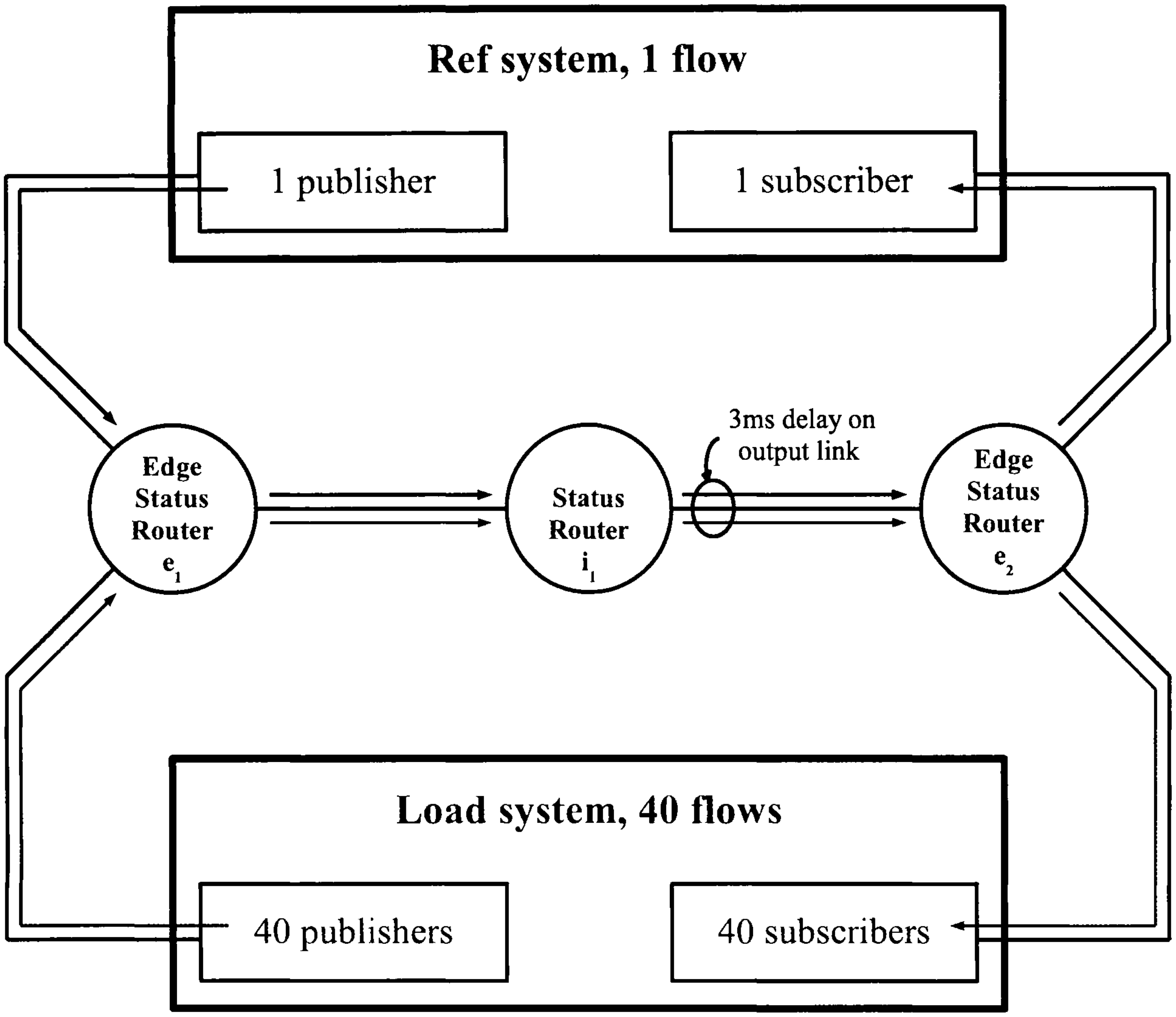


Figure 6.1: Experimental topology

The reference (ref) system is a 1-1 system. That means there is one publisher and one subscriber in the system. The publisher publishes at 50ms intervals and the subscriber has a 50ms interval subscription rate.

The load system is a 40-40 system. That means there are 40 publishers and 40 subscribers in the system. All 40 publishers publish at a 200ms interval and each of the 40 subscribers have a 200ms interval subscription rate.

When in Delay EDD mode for status router i1, d_i the local delay bound is set to 4ms for the ref system and 100ms for the load system. See Appendix B for a derivation of why the local delay bound was set to 4ms for the ref system.

Each experiment was run for 45 minutes. The packets received during the first 15 minutes were not recorded to allow sufficient time for initialization of the Java VM. The Java HotSpot VM is used and its dynamic compilers adaptively compile frequently executed Java bytecodes, or "hot spots", into optimized machine instructions. Therefore, the effective data collection time interval for each experiment is 30 minutes.

On the output link of the internal status router, i1, a 3ms delay was inserted after each packet sent. The objective of this was to simulate a congested link and to cause queuing in the router to occur. To obtain a 3ms delay, the Thread sleep() method was used. Unfortunately this method was highly inaccurate and at times the actual delay ranged from less than 3ms to more than double the 3ms requested delay (see Appendix A).

6.2.1 Experiment 1 – Reference and load systems without 3ms delay

Experiment number one, see figure 6.2, was run to obtain some baseline performance results. Both the reference and load systems were run at the same time, but without the 3ms delay time added after each packet was sent from status router i_1.

Results: The end-to-end delay results for both scheduling algorithms are basically the same. This is not surprising due to the lack of queuing in the status routers. See table 6.1 for a summary of the results.

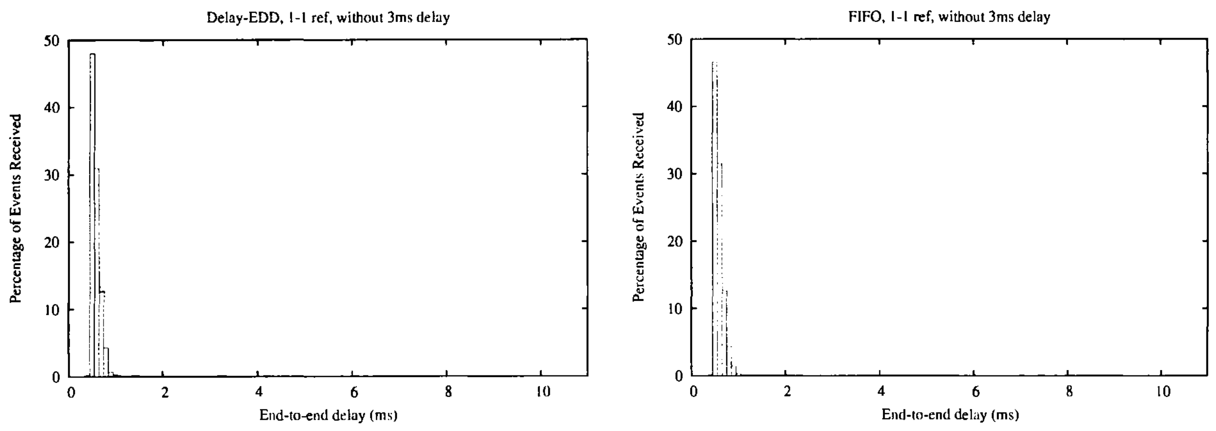


Figure 6.2: Experiment 1 Ref system, Delay-EDD and FIFO

Scheduling Algorithm	System	Min	Mean	Max	Mode	Standard Deviation
Delay-EDD	ref	0.4	0.66	9.1	0.5 (47.9%)	0.58
FIFO	ref	0.4	0.67	10.6	0.5 (46.5%)	0.60

Table 6.1: Experiment 1 Results summary, in milliseconds

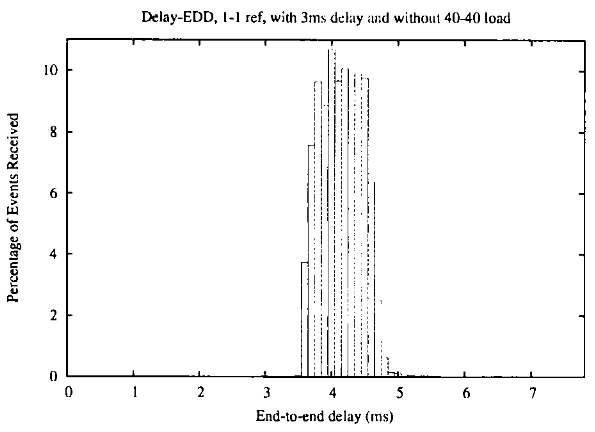
6.2.2 Experiment 2 – Reference system with 3ms delay, but without load system

Experiment number two, see figure 6.3, was run to obtain performance results with the reference system, but without the 40-40 load system added. The 3ms delay time was added after each packet was sent from status router i_1.

Results: The end-to-end delay results for both scheduling algorithms are basically the same. See table 6.2 for a summary of the results. The reason the Delay EDD results

Scheduling Algorithm	System	Min	Mean	Max	Mode	Standard Deviation
Delay-EDD	ref	2.6	4.15	7.5	4.0 (10.68%)	0.33
FIFO	ref	2.7	4.15	7.6	4.0 (10.52%)	0.33

Table 6.2: Experiment 2 Results summary, in milliseconds



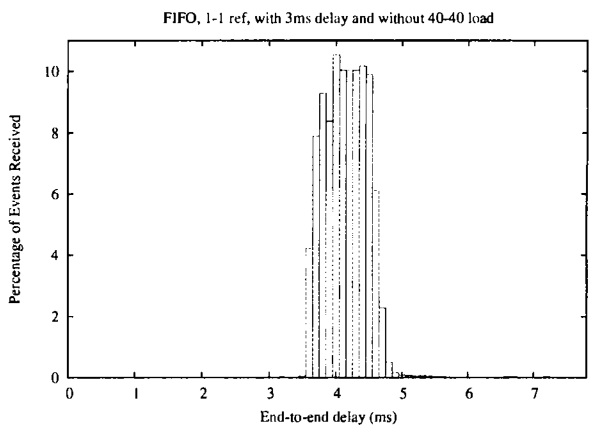


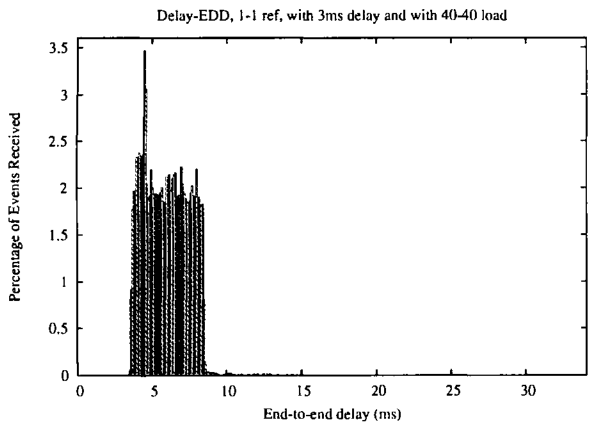
Figure 6.3: Experiment 2 Ref system, Delay-EDD and FIFO

are similiar to the FIFO results is because as the 1-1 ref packets go through the Delay EDD status router they queue up and are serviced in order just as in the FIFO status router. Therefore, the result is as expected that the Delay EDD delays are similiar to the FIFO delays.

6.2.3 Experiment 3 – Reference and load systems with 3ms delay

Experiment number three, see figure 6.4, was run to obtain performance results with the 3ms delay time added after each packet was sent and with the 40-40 load system added. The delay was added to force queuing in the status router and to observe the direct effect of queuing on the end-to-end delay for packets in the 1-1 ref system depending on which scheduling algorithm was being used.

For status router i1, d_i the local delay is increased from 4ms to 17ms for the ref system. This is due to adding a 3ms delay on the output link for each packet sent. Using results from experiment three that max actual sleep time was 6.5ms and a ref packet would have to wait for at most one other packet to be sent before itself being sent, 13ms was added to 4ms (4ms being the max delay in the router without the 3ms delay on the output



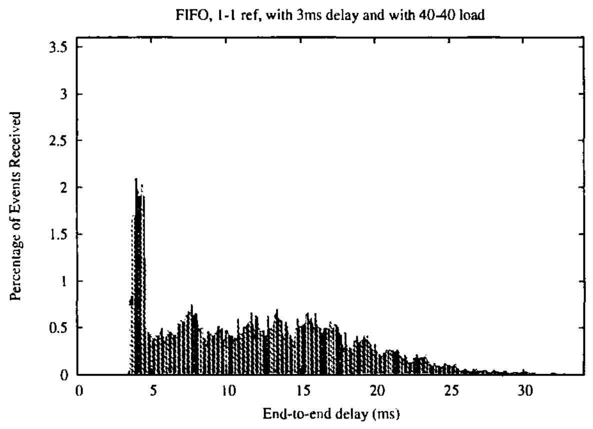


Figure 6.4: Experiment 3 Ref system, Delay-EDD and FIFO

link, see Appendix B) to get the total of 17ms for local delay d_i.

For Delay EDD, both edge status routers local delay bounds were set to 4ms and internal status router set to 17ms. Therefore the end-to-end delay bound for this experiment equals 25ms.

Results: The result was clearly lower end-to-end delays for packets traveling through the Delay EDD router as opposed to the FIFO router. See table 6.3 for a summary of the results. All ref system packets through the Delay EDD status routers achieved their

Scheduling Algorithm	System	Min	Mean	Max	Mode	Standard Deviation
Delay-EDD	ref	2.7	5.99	15.0	4.5 (3.47%)	1.45
FIFO	ref	2.8	11.76	34.1	4.0 (2.09%)	6.15

Table 6.3: Experiment 3 Results summary, in milliseconds

end-to-end delay requirements of 25ms or less. The maximum time taken by a ref packet going through Delay-EDD routers was 15ms. It is observed that with FIFO scheduling, 31.62% of the ref system packets were delivered end-to-end in over 15ms.

6.2.4 Experiment 4 – Local delay at status router i1

Experiment number four, see figure 6.5, was performed to investigate the local delay times experienced at status router i1 under both scheduling algorithms. Both the reference and load systems were run at the same time and the 3ms delay time was added to output of status router i_1. This was an effort to better understand the performance of the internal status router with added delay of 3ms after each packet sent.

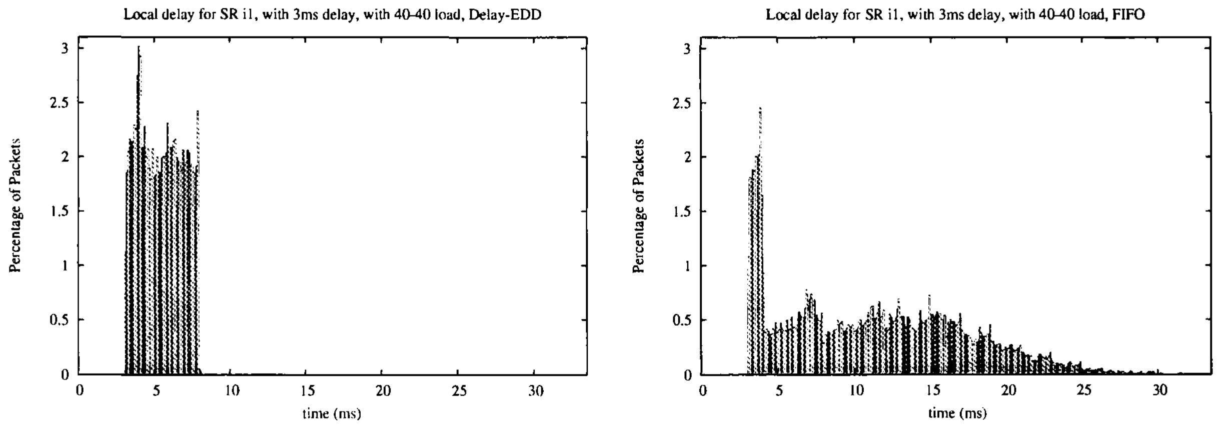


Figure 6.5: Experiment 4 Ref system, Delay-EDD and FIFO, for local SR i_1

Results: Most importantly, with the local delay bound set to 17ms, all ref system packets going through the Delay EDD status router i_1 were sent out before their deadline. See table 6.4 for a summary of the results. The maximum time taken by a ref packet going

Scheduling Algorithm	System	Min	Mean	Max	Mode	Standard Deviation
Delay-EDD	ref	2.1	5.46	10.6	4.0 (3.02%)	1.42
FIFO	ref	2.2	11.13	33.5	3.9 (2.46%)	6.12

Table 6.4: Experiment 4 Results summary, in milliseconds

through Delay-EDD routers was 10.6ms. It is observed that with FIFO scheduling through the status router, 50.47% of the ref system packets were sent out in over 10.6ms.

6.2.5 Experiment 5 – Add an internal status router

The topology for experiment number five, see Figure 6.6, was changed to add one internal status router on the ref system path. To force queuing in status router i_2, edge

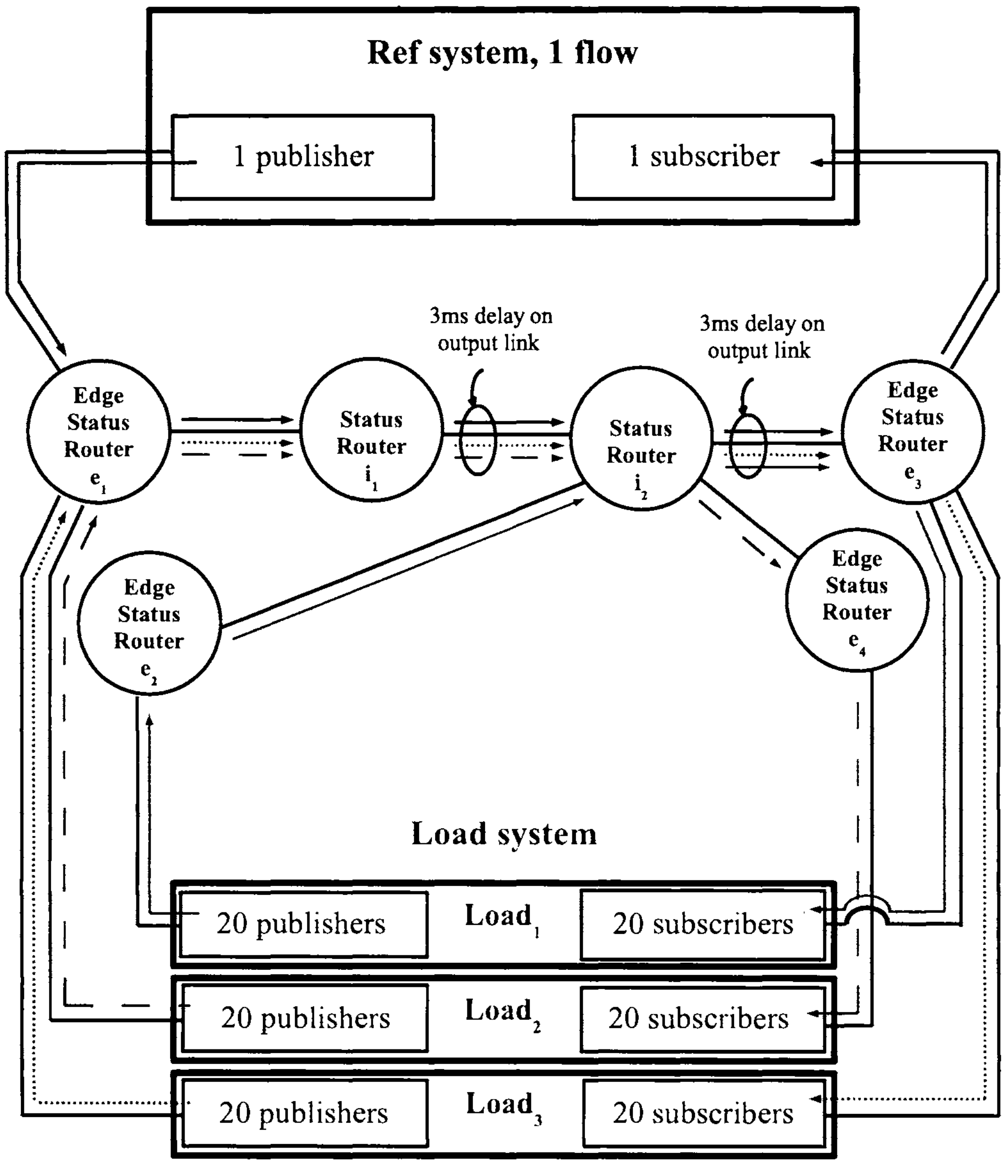


Figure 6.6: Experimental topology

status router e_2 was added to channel 20 (bursty) pub flows to i_2 and edge status router e_4 was added to remove 20 (not bursty) pub flows from i_2. Without this configuration the packets that arrive at i_2 are not bursty, due to status router i_1 sending out one packet every 3ms when there is a packet in the outgoing queue and therefore queuing will not take place in status router i_2 under these conditions.

Both internal status routers experience a 40-40 load system, because at status router i_1 there is incoming $Load_2$ plus $Load_3$ and at status router i_2 there is incoming $Load_1$ plus $Load_3$.

This experiment, see figure 6.7, was done to compare the end-to-end delays under both scheduling algorithms, with an additional status router on the path of the reference system packets. Both the reference and load systems were run at the same time and the 3ms delay time was added to output of both status router i_1 and status router i_2.

For Delay EDD, both edge status routers local delay bounds were set to 4ms and internal status routers set to 17ms. Therefore the end-to-end delay bound for this experiment equals 42ms.

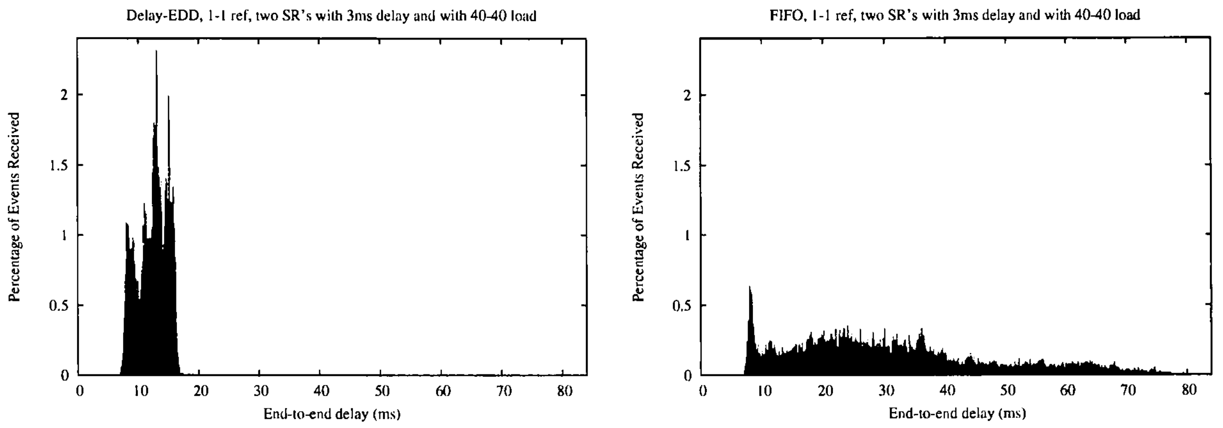


Figure 6.7: Experiment 5 Ref system, Delay-EDD and FIFO, two internal status routers

Results: The result was lower end-to-end delays for packets traveling through the Delay EDD router as opposed to the FIFO router. See table 6.5 for a summary of the results. All ref system packets through the Delay EDD status routers achieved their end-to-end delay requirements of 42ms or less. The maximum time taken by a ref packet

Scheduling Algorithm	System	Min	Mean	Max	Mode	Standard Deviation
Delay-EDD	ref	6.2	12.40	22.9	13.0 (2.32 %)	2.52
FIFO	ref	6.8	31.09	83.1	8.0 (0.64 %)	16.83

Table 6.5: Experiment 4 Results summary, in milliseconds

going end-to-end through Delay-EDD routers was 22.9ms. It is observed that when FIFO scheduling is employed, 63.05% of the ref system packets were delivered end-to-end in over 22.9ms.

6.3 Conclusions from the Experiments

The experiments were performed to compare the end-to-end delay characteristics of the two scheduling algorithms, Delay EDD and FIFO. However, some limitations of the experimental methodology should be noted. It is a limitation of the current status router that queuing due to the network is not actually manageable in Java, as a consequence it was necessary to insert artificial delays to cause queuing to occur. The large variations in inserted delay were not controllable and yet not realistic either.

The results show that overall Delay-EDD performed better than FIFO, under these experimental conditions. With queuing at the router, Delay-EDD will use currently updated deadlines and a sorted queue to always send out the packet with the earliest deadline first.

Experiment three showed a situation where all ref packets through the Delay-EDD status routers were able to achieve their end-to-end delay requirements of 25ms with a maximum of 15ms. Whereas in the same situation, under FIFO scheduling, many ref system packets were delivered end-to-end in over 25ms with a maximum of 34.1 ms. When more than one packet is in the queue, Delay-EDD can make a distinction between packets in the queue and send the one with the earliest deadline first. Even with end-to-end delay requirements of 25ms, the Delay-EDD status routers delivered all packets in less than 15ms. This demonstrates that although worst-case bounds are allowed for at each hop, it is unlikely that a packet would ever experience a sequence of worst-case delay events at all hops on the path from source to destination.

By factoring in the actual sleep times from Appendix A, the local delay times experienced at status router i_1 can be better understood. As packets are being added to the sorted queue, each packet from the ref system is delayed not only when it gets sent on the outgoing channel, but also in waiting for the current packet undergoing transmission to be sent.

Experiment five showed that when adding one additional internal status router on the ref system path, all of the ref packets going through the Delay-EDD status routers were able to achieve their end-to-end delay requirements of 42ms with a maximum of 22.9ms. This was not unexpected because adding the maximum end-to-end delay seen in experiment three of 15ms to the maximum local delay of 10.6ms from experiment four, equals 25.6ms. So, it would be expected for the maximum end-to-end delay for a ref packet to be less than 25.6ms in experiment five and indeed the maximum end-to-end delay for the ref packets through Delay-EDD was 22.9ms.

The problem with FIFO scheduling, as can be observed most easily on the 1-1 reference system, is that packets that arrive on a congested link must wait in line to be serviced no matter what. No distinction between packets is made and therefore packets needing a short delay can be held up by packets that could take a longer delay.

Chapter 7
Conclusion

In this thesis, the problem of bounding end-to-end delay in a real-time status dissemination network, specifically GridStat, for multiple traffic flows routed over dedicated lines was addressed. GridStat is a status dissemination middleware framework that is being developed for the electric power grid. It has a number of features that make it easier to provide delay guarantees. Static route packet switching, as opposed to dynamic routing, is used exclusively. Jitter is not an issue, only that end-to-end delay is bounded. It makes use of a dedicated network for real-time traffic, so traffic allowed onto the network can be tightly controlled.

It was shown that simple scheduling algorithms in the routers, like first in first out or priority are not adequate, because they do not protect well behaving flows from different sources of variability inside the network. Existing guaranteed rate scheduling algorithms, that do offer protection, were identificated and analysis was done to determine the most appropriate one to use for GridStat. Although the use of *any* guaranteed rate scheduling algorithm in the status routers would have sufficed to attain the desired aim of bounded end-to-end delay.

After evaluating the pros and cons of all the work-conserving scheduling algorithms, it was determined that the most appropriate scheduling algorithm for GridStat is Delay-Earliest Due Date (Delay-EDD). Implementation of the Delay-EDD scheduling algorithm in a GridStat prototype was done as a proof of concept. Based on the experimental results obtained it can be shown that a Delay-EDD router can deliver bounded end-to-end delay in cases where a FIFO router cannot.

In summary, in a status dissemination network like GridStat, with traffic flows that travel on a dedicated network, the use of admission control, resource reservation, and guaranteed rate scheduling algorithm will achieve bounded end-to-end delay for admitted traffic flows that obey their traffic specifications.

7.1 Future Work

Since two disjoint paths are used for every publisher-subscriber pair, do both of them really need real-time performance guarantees? It might be possible that a hybrid approach could be set up to accommodate split performance guarantees. One disjoint path could be real-time and the other best effort, for example. The advantage would be increased network utilization, but at the expense of opening up the possibility for lost or delayed packets on the backup path.

Even though GridStat makes use of a dedicated network and rate control in the routers, misbehaving traffic flows could still occur within the network. In the event of a cyber attack on the system, it is conceivable that status routers could be overloaded with manufactured status or alert packets that are not genuine. This could compromise the end-to-end delay guarantees of existing well-behaving flows. Therefore, a Run-Time Monitoring System to monitor for bogus packets and misbehaving flows could be added to GridStat.

Bibliography

[And00] Matthew Andrews. Probabilistic End-to-End Delay Bounds for Earliest Deadline First Scheduling. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 603–612, 2000.

[BBC$^+$98] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An Architecture for Differentiated Service. IETF RFC 2475, December 1998.

[BBH$^+$02] David Bakken, Anjan Bose, Carl Hauser, Ioanna Dionysiou, Harald Gjermundrød, Lin Xu, and Sudipto Bhowmik. Towards More Extensible and Resilient Real-Time Information Dissemination for the Electric Power Grid. In *Power Systems and Communications Systems for the Future*, International Institute for Critical Infrastructures, Beijing, September 2002.

[BCS94] Robert Braden, David Clark, and Scott Shenker. Integrated Services in the Internet Architecture: an Overview. IETF RFC 1633, June 1994.

[BFY$^+$00] Yoram Bernet, Peter Ford, Raj Yavatkar, Fred Baker, Lixia Zhang, Michael Speer, Robert Braden, Bruce Davie, John Wroclawski, and Eyal Felstaine. A Framework for Integrated Services Operation over Diffserv Networks. IETF RFC 2998, November 2000.

[BZ96] Jon C. R. Bennett and Hui Zhang. WF^{2}Q: Worst-Case Fair Weighted Fair Queueing. In *Proceedings of IEEE INFOCOM '96*, pages 120–128, March 1996.

[CG89] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.

[CL02] Nicolas Christin and Jörg Liebeherr. Providing Strong Service Guarantees with a Scalable Service Architecture. In *Scalability and Traffic Control in IP networks II*. Proceedings of SPIE, volume 4868, pages 31–42, Boston, MA, 2002.

[Cru91a] Rene L. Cruz. A Calculus for Network Delay. I. Network Elements in Isolation. *IEEE Transactions on Information Theory*, 37:114–131, 1991.

[Cru91b] Rene L. Cruz. A Calculus for Network Delay. II. Network Analysis. *IEEE Transactions on Information Theory*, 37:132–141, 1991.

[DKS89] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *SIGCOMM '89: Symposium Proceedings on Communications Architectures & Protocols*, pages 1–12. ACM Press, 1989.

[EM99] Anwar Elwalid and Debasis Mitra. Design of Generalized Processor Sharing Schedulers Which Statistically Multiplex Heterogeneous QoS Classes. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1220–1230, 1999.

[Fer90] Domenico Ferrari. Client Requirements for Real-Time Communication Services. *IEEE Communications Magazine*, 28(11):65–72, 1990.

[FP95] Norival R. Figueira and Joseph Pasquale. An Upper Bound on Delay for the VirtualClock Service Discipline. *IEEE/ACM Transactions on Networking*, 3(4):399–408, 1995.

[FV90] Domenico Ferrari and Dinesh C. Verma. A Scheme for Real-Time Channel Establishment in Wide-Area Networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, 1990.

[GDB$^+$03] Kjell Harald Gjermundrød, Ioanna Dionysiou, David Bakken, Carl Hauser, and Anjan Bose. Flexible and Robust Status Dissemination Middleware for the Electric Power Grid. Technical report, Washington State University, 2003.

[Gje06] Kjell Harald Gjermundrød. *Flexible QoS-Managed Status Dissemination Middleware Framework for the Electric Power Grid.* PhD thesis, Washington State University (WSU), 2006.

[GLV97] Pawan Goyal, Simon S. Lam, and Harrick M. Vin. Determining End-to-End Delay Bounds in Heterogeneous Networks. *Multimedia Syst.*, 5(3):157–163, 1997.

[Gol90] S. Jamaloddin Golestani. A Stop-and-Go Queueing Framework for Congestion Management. In *SIGCOMM '90: Proceedings of the ACM symposium on Communications Architectures & Protocols*, pages 8–18, 1990.

[Gol94] S. Jamaloddin Golestani. A Self-Clocked Fair Queueing Scheme for Broadband Applications. In *Proceedings of the IEEE INFOCOM '94*, pages 636–646, June 1994.

[GV97] Pawan Goyal and Harrick M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. *IEEE/ACM Transactions on Networking*, 5(4):561–571, 1997.

[HBB05] Carl H. Hauser, David E. Bakken, and Anjan Bose. A Failure to Communicate: Next Generation Communication Requirements, Technologies, and Architecture for the Electric Power Grid. *IEEE Power and Energy Magazine*, 3:47–55, 2005.

[HBD+07] Carl H. Hauser, David E. Bakken, Ioanna Dionysiou, K. Harald Gjermundrød, Venkata S. Irava, Joel Helkey, and Anjan Bose. Security, trust and QoS in next-generation control and communication for large power systems. *International Journal of Critical Infrastructures*, to appear 2007.

[HLP91] Jay M. Hyman, Aurel A. Lazar, and Giovanni Pacifici. Real-Time Scheduling with Quality of Service Constraints. *IEEE Journal of Selected Areas in Communications*, 9(7):1052–1063, 1991.

[Ira06] Venkata S. Irava. *Low-cost delay-constrained multicast routing heuristics and their evaluation.* PhD thesis, Washington State University (WSU), 2006.

[Kes97] Srinivsan Keshav. *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network.* Addison-Wesley Professional, 1st edition, 1997.

[KKK90] Charles R. Kalmanek, Hemant Kanakia, and Srinivasan Keshav. Rate Controlled Servers for Very High-Speed Networks. In *Proceedings of the Conference on Global Communications (GLOBECOM)*, December 1990.

[KR05] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison Wesley, third edition, 2005.

[KSF91] Dilip D. Kandlur, Kang G. Shin, and Domenico Ferrari. Real-Time Communication in Multi-Hop Networks. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, Arlington, TX, May 1991.

[Kur92] Jim Kurose. On computing per-session performance bounds in high-speed multi-hop computer networks. In *SIGMETRICS '92/PERFORMANCE '92:*

Proceedings of the 1992 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, pages 128–139, New York, NY, USA, 1992. ACM Press.

[LPY01] Jörg Liebeherr, Stephen D. Patek, and Erhan Yilmaz. Tradeoffs in Designing Networks with End-to-End Statistical QoS Guarantees. Technical Report CS-2001-11, University of Virginia, Department of Computer Science, February 2001.

[LX95] Simon S. Lam and Geoffrey G. Xie. Burst Scheduling: Architecture and Algorithm for Switching Packet Video. In *Proceedings of INFOCOM '95*, pages 940–950, 1995.

[MA99] John Mountford and Ricardo Austria. Keeping the lights on [power system reliability]. *IEEE Spectrum*, 36:34–39, 1999.

[Mok83] Aloysius K. Mok. *Fundamental design problems of distributed systems for the hard real-time environment.* PhD thesis, Massachusetts Institute of Technology (MIT), 1983.

[OPSS93] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The information bus: an architecture for extensible distributed systems. In *SOSP '93: Proceedings of the fourteenth ACM symposium on Operating Systems principles*, pages 58–68. ACM Press, 1993.

[Par92] Abhay K. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks.* PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1992.

[PG92] Abhay K. Parekh and Robert G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks - The Single Node Case. In *Proceedings of the INFOCOM '92*, pages 915–924, 1992.

[PG93] Abhay K. Parekh and Robert G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. In *Proceedings of the INFOCOM '93*, pages 521–530, 1993.

[Puz02] Rita Puzmanova. *Routing and Switching: Time of Convergence?* Addison-Wesley, 2002.

[RRR98] Martin Reisslein, Keith W. Ross, and Srinivas Rajagopal. Guaranteeing Statistical QoS to Regulated Traffic: The Multiple Node Case . In *IEEE Conference on Decision and Control, 1998*, volume 1, pages 531–538, 1998.

[RRR99] Martin Reisslein, Keith W. Ross, and Srinivas Rajagopal. Guaranteeing Statistical QoS to Regulated Traffic: The Single Node Case. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1061–1072, 1999.

[RRR02] Martin Reisslein, Keith W. Ross, and Srinivas Rajagopal. A framework for guaranteeing statistical qos. *IEEE/ACM Trans. Netw.*, 10(1):27–42, 2002.

[SC99] Vijay Sivaraman and Fabio M. Chiussi. Statistical Analysis of Delay Bound Violations at an Earliest Deadline First (EDF) Scheduler. *Performance Evaluation*, 36-37(1-4):457–470, 1999.

[SLDZ97] Anirudh Sahoo, Chengzhi Li, Badari Devalla, and Wei Zhao. Design and Implementation of NetEx: A Toolkit for Delay Guaranteed Communications. In *Proceedings of Military Communications Conference (MILCOM)*, November 1997.

[SPG97] Scott Shenker, Craig Partridge, and Roch Guerin. Specification of Guaranteed Quality of Service. IETF RFC 2212, September 1997.

[Sta88] John A. Stankovic. Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems. *Computer*, 21(10):10–19, 1988.

[TBVB05] Kevin Tomsovic, David Bakken, Vaithianathan Venkatasubramanian, and Anjan Bose. Designing the Next Generation of Real-Time Control, Communication and Computations for Large Power Systems. In *Proceedings of the IEEE (Special Issue on Energy Infrastructure Systems)*, volume 93, pages 965–979, 2005.

[VZF91] Dinesh Verma, Hui Zhang, and Domenico Ferrari. Guaranteeing Delay Jitter Bounds in Packet Switching Networks. In *Proceedings of TriComm '91*, pages 35–46, Chapel Hill, North Carolina, April 1991.

[Wro97a] John Wroclawski. Specification of the Controlled-Load Network Element Service. IETF RFC 2211, September 1997.

[Wro97b] John Wroclawski. The Use of RSVP with IETF Integrated Services. IETF RFC 2210, September 1997.

[Zha90] Lixia Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet-Switched Networks. *SIGCOMM ACM Sumposium*, pages 19–29, 1990.

[Zha95] Hui Zhang. Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. In *IEEE*, volume 83, pages 1374–1396, Oct 1995.

[ZK91] Hui Zhang and Srinivasan Keshav. Comparison of Rate-Based Service Disciplines. In *SIGCOMM*, pages 113–121, 1991.

[ZK96] Hui Zhang and Edward W. Knightly. RCSP and Stop-and-Go: A Comparison of Two Non-Work-Conserving Disciplines for Supporting Multimedia Communication. *Multimedia Systems*, 4(6):346–356, 1996.

[ZS94] Qin Zheng and Kang G. Shin. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks. *IEEE Transactions on Communications*, pages 1096–1105, March 1994.

Appendix A
Actual sleep time

In modifying the GridStat prototype, to obtain a 3ms delay on the output link of status router i_1, the Java Thread sleep() method was used to cause the current thread to suspend execution for a specified 3ms. However, the Thread sleep() times are not guaranteed to be precise, therefore it cannot be assumed that invoking sleep() will suspend the thread for precisely the time period specified.

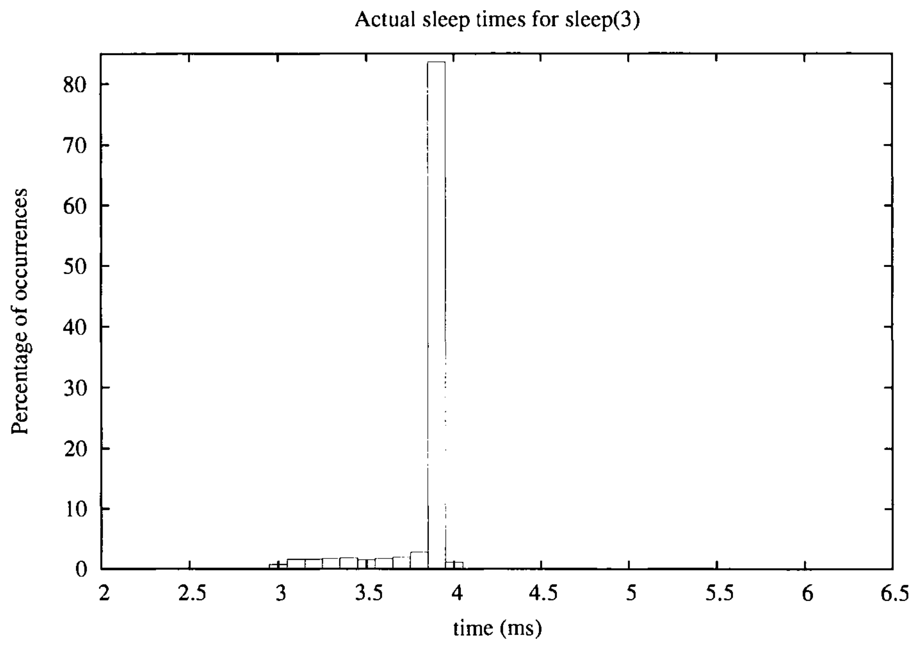


Figure A.1: Actual sleep times for requested sleep(3) on SR i1

This experiment was run to characterize the actual sleep times returned by sleep(3), a requested 3 millisecond sleep. Figure A.1 shows a graph of the actual sleep times obtained. The time duration for data logging was 30 minutes and all Thread.sleep() *method* calls were recorded during that time period.

	Min	Max	Mean	Standard Deviation
Actual recorded sleep times (ms)	2.0	6.5	3.84	0.19

Table A.1: Actual sleep times for requested 3ms delay

Results: See table A.1 for a summary of the results. The actual sleep times are profiled in table A.1. The mode was 3.9ms (83.62%). Total number of sleep() method calls during the data logging time was $\approx 456,000$. % Appendix

Appendix B
SR i_1 local delay bound derivation

The local delay bound derivation for SR i_1 was based on ref system experimental data without 3ms delay of packet departures from SR i_1 minus packet arrivals at SR i_1. First, the 3ms delay on the output link of SR i_1 was removed. Then, the reference system was run both without and with the 40-40 load system for 45 minutes each (15 minutes for settle down time and then 30 minutes duration for data recording), see table B.1. Each test run had a total of 40,394 reference packets travel through the status router.

The packet that was delayed the most without the load system had a 2.9ms delay. The packet that was delayed the most with the load system had a 3.3ms delay. The maximum of the two (3.3ms) was rounded up to the nearest millisecond. Therefore, the experimentally derived local delay bound for SR i_1 was determinded to be 4 ms.

The data distribution exhibits a long tail both with and without the load system. Consequently, the 4ms local delay bound represents an explicit worst case as greater than 98% of the packets are handled in less than 0.2ms.

time (ms)	without 40-40 load	with 40-40 load
0.0	79.1578%	95.0488%
0.1	19.0127%	4.4833%
0.2	1.5844%	0.2451%
0.3	0.0421%	0.0124%
0.4	0.0173%	0.0149%
0.5	0.1139%	0.0891%
0.6	0.0124%	0.0272%
0.7	0.0000%	0.0050%
0.8	0.0050%	0.0000%
0.9	0.0050%	0.0099%
1.0	0.0025%	0.0074%
1.1	0.0000%	0.0000%
1.2	0.0000%	0.0000%
1.3	0.0000%	0.0000%
1.4	0.0000%	0.0000%
1.5	0.0000%	0.0000%
1.6	0.0000%	0.0000%
1.7	0.0000%	0.0000%
1.8	0.0000%	0.0000%
1.9	0.0000%	0.0000%
2.0	0.0000%	0.0000%
2.1	0.0000%	0.0000%
2.2	0.0000%	0.0000%
2.3	0.0000%	0.0000%
2.4	0.0000%	0.0000%
2.5	0.0000%	0.0000%
2.6	0.0000%	0.0000%
2.7	0.0099%	0.0025%
2.8	0.0297%	0.0198%
2.9	0.0074%	0.0198%
3.0	0.0000%	0.0124%
3.1	0.0000%	0.0000%
3.2	0.0000%	0.0000%
3.3	0.0000%	0.0025%
3.4	0.0000%	0.0000%
3.5	0.0000%	0.0000%

Table B.1: SR i_1 Local delay without 3ms delay