Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

MODELLING DIFFERENTIATED SERVICES IN CONSERVATIVE PDES

Roger Curry
Rob Simmonds
Brian Unger

Department of Computer Science
University of Calgary
Calgary, Alberta, T2N 1N4, CANADA

ABSTRACT

This paper explains how DiffServ has been implemented in an IP network simulator using an asynchronous conservative parallel discrete event simulation (PDES) kernel. DiffServ provides Quality of Service (QoS) functionality for IP networks and is designed to provide greater scalability and lower overhead than previous IP based QoS schemes. The paper explains the DiffServ components that have been implemented, focusing on the implementation of the preemptive network buffers required to provide DiffServ functionality. Certain optimisations possible for non-preemptive network buffers are not possible here. The paper explores which will work in the preemptive case. In particular, exploiting lookahead is more difficult leading to reduced performance in some cases. Optimisation schemes are described for two different preemptive buffering strategies and performance results demonstrating the costs of using these buffers are presented.

1 INTRODUCTION

This paper explains some of the modelling techniques used in implementing DiffServ in the IP-TN/E parallel discrete event network simulation and emulation system (Simmonds, Bradford, and Unger 2000). In particular, it explains the different network buffer implementations that already exist in the system and the new ones added to support DiffServ. Network buffers are used by hosts and routers to hold packets that are waiting to be written out to a network link. We examine the difference between the implementation of buffers that support non-preemptive strategies and ones that support preemptive strategies. Preemptive strategies are required to model some of the functionality provided by DiffServ enabled routers (Cisco 2003).

DiffServ provides end-to-end quality of service (QoS) functionality on IP networks. It is designed to provide greater scalability than previous QoS mechanisms such as IntServ and RSVP (see RFC 2210). The aim in deploying DiffServ is to meet the requirements of different types of traffic traversing a network segment. For example, voice traffic using the voice over IP protocol (VoIP) requires little bandwidth but requires low latency and jitter. On the other hand, the File Transfer Protocol (FTP) requires large bandwidth, but can tolerate large latencies, jitter and some packet loss.

The Internet Protocol Traffic and Network (IP-TN) simulator and the IP-TNE network emulation system that shares the same core simulation components use a channel oriented logical process modelling methodology (Chandy and Misra 1981, Bryant 1977). The CCTKit conservative PDES kernel (Simmonds, Kiddle, and Unger 2002) is used. This has been shown to provide high performance sequential operation as well as high performance parallel operation on shared memory parallel computers. CCTKit uses task partitioning and schedules tasks using the Critical Channel Traversing (CCT) algorithm (Xiao et al. 1999).

The addition of DiffServ functionality to the IP-TN/E system has been done in order to support simulation and emulation studies being performed in cooperation with the company developing the Alberta SuperNet. SuperNet will eventually contain over 8410km of fibre optic cable divided into two distinct networks: the base area network, and the extended area network. Axia SuperNet Ltd. is responsible for building the extended area network and for the operation and management of the entire Alberta SuperNet once it is completed. IP-TN has to be able to model the DiffServ functionality provided by specific Cisco routers, therefore the IP-TN/E DiffServ implementation has been based on functionality provided by Cisco's Internet Operating System (IOS).

SuperNet will provide broadband access to all schools, colleges, hospitals, clinics, libraries and government organisations in Alberta. Particular projects of interest include remote learning using streaming media and collaborative visualisation for both schools and medical practitioners. Latency sensitive traffic sources will have to compete with all of the other traffic seen today on general use networks, i.e., HTTP downloads, peer-to-peer file sharing, etc.

Before DiffServ was added to IP-TN, the only network buffer implementations in IP-TN were non-preemptive. With these, packets are removed from each network buffer in first-in, first-out (FIFO) order. This correctly models most network buffers, but DiffServ requires various forms of preemptive service. One strategy forwards packets strictly according to their marked priority. Another strategy involves sharing bandwidth according to marked priority. Both of these strategies require different buffer implementations and offer different opportunities for optimised modelling. DiffServ also requires support for preferentially dropping packets, although this can be accomplished without preventing any of the buffering optimisations.

The rest of the paper is as follows. Section 2 gives an overview of DiffServ. Section 3 describes the IP-TN simulator, gives a brief description of the CCTKit parallel simulation kernel used by IP-TN and describes the DiffServ functionality that has been implemented. Section 4 describes the implementation of the various network buffers in IP-TN. Then section 5 presents results comparing the performance of simulations that use the different network buffer types implemented in IP-TN/E. Section 6 provides a summary and concluding remarks.

2 DIFFSERV

This section describes Differentiated Services (DiffServ) and how DiffServ functionality is provided in router's using Cisco IOS.

DiffServ is a quality of service mechanism for IP networks. It was designed to overcome some of the overhead and scaling problems associated with the previous generation IP network QoS mechanisms such as IntServ and RSVP. IntServ provided QoS on a per-flow basis, while DiffServ distributes network resources among classes of traffic.

The Differentiated Services Code Point (DSCP) is a 6-bit field in the IP header used to select the Per Hop Behaviour (PHB) for the packet at each network node. This field is a redefinition of the IPv4 Type of Service (TOS) 3-bit precedence field. DiffServ's assured forwarding (AF) classes are defined in RFC 2597 (Heinanen, et al. 1999). Within each AF class (AF1x through AF4x) there are three drop probabilities "low," "medium," and "high"; however the DiffServ standard does not specify a precise definition of these. The Expedited Forwarding (EF) PHB is defined in RFC 2598 (Jacobsen, Nichols, and Poduri 1999). EF is to provide low loss, low latency, low jitter, assured bandwidth through a DiffServ domain. Table 1 shows the different DSCP values and their associated PHBs.

Packets are marked with a DSCP suitable for what is required by their application and to fit in with the policies of an organisation. For example, it could be that all VoIP traffic from particular addresses are marked EF. Similarly traffic from collaborative visualisation rooms may be marked EF. Traffic from less important video streaming source may be marked with AF43, to indicate that they want low latency (hence using AF4x) but are prepared to be dropped if there is not enough space in the network to give low latency operation. Streams performing file-system backups across a network may work fine with high latencies, but work best if packets are not dropped. Therefore a DSCP of AF11 may be selected for these packets. It is also possible to have different types of packets in the same stream marked differently. For example good performance has been shown for TCP when routers give preference to certain packets (Williamson and Wu 2002). In particular if some packets near the end of a connection are lost it can be far more harmful to the overall performance than if a packet near the start of a data transfer is lost. Therefore different DSCP values could be given to different packets to reflect this.

The marking of packets could be done by applications themselves, or by a switch or router on a LAN. Often packets are remarked by the network provider to fit a Service Level Agreement (SLA) between the owner of the LAN and the network provider. If the packets are marked on the LAN it is possible that they could be remarked by the network provider if the LAN is outputting packets marked in a way that does not agree with the SLA. In particular the SLA may state that only a certain bandwidth of high priority traffic will be carried. A traffic policing mechanism could be used to relabel packets when these limits are exceeded. Note that to avoid packet reordering it is likely that the policer would try to increase the drop probability rather than change the AF class if this is possible.

Routers in a DiffServ domain (network) may be either edge or core routers. In core routers, the forwarding behaviour for a given packet is determined by its DSCP field. Edge routers are referred to as ingress or egress routers depending on whether packets are entering or leaving the DiffServ domain. In addition to forwarding packets, ingress routers are also used to shape and police different traffic flows. Ingress routers often aggregate existing flows or remark the DSCP field as appropriate for the particular DS domain.

Cisco's IOS provides several mechanisms to implement packet forwarding policies in each network buffer. Two such mechanisms are *Priority Queuing* and *Custom Queuing*. Priority Queuing is implemented using four FIFO queues. The network administrator sets up a table that maps packets with each possible DSCP value to a particular queue. Traffic mapped to the first queue is treated with the highest priority, so if there is a packet ready to be sent from this queue when the network device is ready to write to a link, the first packet from this highest priority queue will be sent. Each of the remaining queues are only serviced if the higher priority queues are empty. The Priority Queuing mechanism has the

Table 1: DiffServ Codepoint

	SBE	Class 1	Class 2	Class 3	Class 4	EF
Low Drop		001010 AF11 DSCP 10	010010 AF21 DSCP 18	011010 AF31 DSCP 26	100010 AF41 DSCP 34	
Medium Drop	000000 SBE DSCP00	001100 AF12 DSCP 12	010100 AF22 DSCP 20	011100 AF32 DSCP 28	100100 AF42 DSCP 36	101110 EF DSCP 42
High Drop		001110 AF13 DSCP 14	010110 AF23 DSCP 22	011110 AF33 DSCP 30	100110 AF43 DSCP 38	

problem that lower priority queues could suffer starvation in networks carrying large amounts of higher priority packets. Therefore a policing mechanism may need to be configured to prevent this from occurring.

In Custom Queuing packets can be mapped to any of sixteen FIFO queues. The Custom Queuing mechanism works by allocating a certain amount of the currently available link bandwidth to each queue. For example, in a simple configuration you could map packets to four queues. For each queue you can specify the amount of the total buffer space available that will be assigned to this particular queue and total number of bytes that will be serviced from this queue before the scheduler moves to the next queue using a simple round-robin approach.

With both Priority Queuing and Custom Queuing, the network administrator can set the size of each FIFO queue. This is an important part of the configuration since the size of each buffer can play a key role in the way traffic is handled. For example, for packets that require low latency but can be dropped, a small buffer is appropriate. If a large buffer filled up the packets would not meet their latency requirement due to the increased queuing delay. Therefore it is better just to drop packets when too many of this type of packet are waiting. On the other hand, for queues holding packets that can tolerate high latency, making the buffers larger helps to prevent packet loss.

For applications using TCP or other transport layer protocols which respond to packet loss, Random Early Detect (RED) can be enabled as part of a DiffServ strategy that reduces the chance of severe congestion in a preventative, rather than responsive manner. RED avoids the global synchronisation problem that can occur when many TCP flows are repeatedly congesting the network and restarting. IP-TN's DiffServ supports Weighted RED or WRED which is similar to normal RED but takes the DSCP into account when deciding whether or not to drop a packet. For example, separate RED parameters may be specified for each PHB.

3 IP-TN

The Internet Protocol Traffic and Network (IP-TN) simulator (Simmonds, Bradford, and Unger 2000) is a discrete event IP layer network simulator developed by the TeleSim group at the University of Calgary. IP-TN includes modelling support for IPv4, ICMP, IP broadcast and now DiffServ. It also supports fluid-flow and hybrid fluid-packet models, which require special implementations of non-preemptive network buffers (Kiddle, Simmonds, and Williamson 2003); these will not be discussed in this paper. IP-TN forms the basis of the Internet Protocol Traffic and Network Emulator (IP-TNE) which provides a real-time interactive test environment for real network enabled applications and network services. Due to the need to appear correct to real network services, support for protocols such as the Internet Control Message Protocol (ICMP) is more complete than that found in many other network simulation systems.

IP-TN/E uses the CCTKit PDES kernel (Simmonds, Kiddle, and Unger 2002). CCTKit is an asynchronous channel based conservative simulation kernel that implements the Critical Channel Traversing (CCT) synchronisation and scheduling algorithm (Xiao et al. 1999). CCTKit was designed to provide fast parallel execution for a large class of simulation models, but has also proved to be effective for sequential simulation, often outperforming the best purely sequential kernels. The CCT algorithm is under further development by at least two groups (Nicol and Liu 2001; Simmonds, Kiddle, and Unger 2002; Nicol 2002).

CCTKit also uses the task scheduling ideas first introduced in TasKit (Xiao et al. 1999). In this algorithm, logical processes (LPs) are grouped into tasks, with the tasks being scheduled using CCT and the LPs within the task being scheduled in a way most suitable for the connectivity of those LPs. This could be done using a single priority queue, or using a fixed schedule for cases where there are well defined relationships between the LPs in the task. In the case of IP-TN, the priority queue technique is usually employed since the topology of IP networks does not easily lend itself to fixed-schedule execution. CCTKit has been designed to avoid the use of locks wherever possible which leads to considerable performance advantage on many shared memory computers.

As in other IP level network simulators, packets are carried through the network by events. An event is used to represent the arrival of a packet at a host or router. In some cases other events are used to represent packets being added to network buffers or leaving network buffers. How this is done for the various types of network buffers implemented in IP-TN is explained in section 4.

Events are also used to generate traffic and trigger timeouts in hosts and routers. IP-TN has a number of different traffic models including open loop models that send traffic at a constant rate or using various statistical distributions. Also, it has closed loop traffic such as TCP Reno and TCP SACK which can be used with a number of different traffic models including web browser and web server models. The capabilities of the HTTP 1.1 web browser model have been demonstrated for testing large real web servers using IP-TNE (Williamson, Simmonds, and Arlitt 2002).

The DiffServ implementation that has been added to IP-TN/E is designed to model the functionality provided in Cisco's IOS software. As such, it is mostly compliant with IETF standards defined in RFC 2474 (Nichols, et al. 1998), RFC 2475 (Blake, et al. 1999), RFC 2597 and RFC 2598. The various components of IP-TN's DiffServ are now outlined.

IP-TN's DiffServ implementation allows classification of traffic based on IP source and destination addresses, interface, and DSCP value. The modeller may also "classify" packets simply by assigning DSCP codes on a per traffic object basis. Packets can be marked with any of the DSCP values in Table 1.

To facilitate traffic conditioning, IP-TN can also classify packets based on whether they conform to a particular service level agreement (traffic policing). IP-TN uses a leaky-bucket traffic policing algorithm, different actions may be specified for traffic that conforms with or that exceeds the prescribed bandwidth allotment.

Per-hop behaviours can be defined using a combination of priority queuing, custom queuing and, Weighted Random Early Detection (WRED). Per-hop behaviours are selected based on the packet header's DSCP value.

Implementation details of the various congestion management schemes (priority queuing and custom queuing) are presented in the following section which is devoted solely to buffer implementation issues.

4 IP-TN NETWORK BUFFERS

This section describes the implementation of the various network buffers in IP-TN. It starts by describing the three non-preemptive FIFO buffer implementations, followed by the new preemptive buffer implementations that provide DiffServ modelling support.

All IP-TN/E network buffers are modelled as fixed size buffers, the size of which can be defined in the simulation input file. The size of the buffer is defined in terms of number of bytes, rather than number of packets, as with some other network simulators. Unlike ATM cells, IP packets have a variable length between 20 bytes and the Maximum Transfer Unit (MTU) which represents the maximum frame size on the local network.

Some of the buffer implementations attempt to dynamically increase the amount of lookahead which is the amount of time into the future that the occurrence of events at an LP can be predicted. By dispatching events carrying packets to the next host early this increases its knowledge of the future which can lead to increased runtime performance, particularly when executing in parallel.

4.1 Non-Preemptive Buffers

IP-TN/E has three implementations of non-preemptive FIFO buffers called *simple_no_opt*, *standard_opt* and *nolist_opt*.

The simple_no_opt buffers are provided as an easily understandable buffer implementation that is simple for non-PDES experts to modify. They were originally provided to enable testing of context-aware TCP (Williamson and Wu 2002). Later work in the same area builds on the new buffer implementations presented later in this section. The simple_no_opt buffer does not perform any lookahead optimisation.

Actions involved with routing a packet are shown in Figure 1. A packet arrives at a router carried with an *arrival* event. There is some amount of processing and router lookup delay after which an attempt is made to put the packet in the appropriate network buffer. This is a *buffer insert* event. Note that there may not be room to insert the packet into the buffer. In this case the packet would be dropped. If there are other packets waiting to be sent the packet would then remain in a queue until a *check queue* event occurs when it is time to dispatch the packet. After the amount of time taken to pass the packet to the other end of the link passes, which is equal to the propagation delay plus the time taken to transmit the packet across the network, an *arrival* event is triggered at the destination host.

With simple_no_opt buffers, once an arrival event is received a buffer insert (BI) event is created and sent to the current LP; i.e., this is a self event. On receiving a BI event a check queue (CQ) self event is generated if the buffer is currently empty. Otherwise nothing needs to be done at this time since another CQ event has already been posted. When a CQ event is received the next packet waiting to be sent is taken from the buffer and attached to an arrival event that is dispatched to the destination LP. Also, if the buffer is not empty at this point a new CQ event is posted to arrive when the packet just sent will have left the buffer.

The standard_opt buffers utilise a lookahead optimisation. They also take advantage of being non-preemptive by eliminating the BI event. This can be done by either assuming that all router lookups will take the same amount of time, which is the assumption made here, or by assuming that the device doing the router lookup is executing sequentially. When assuming a variable lookup time and

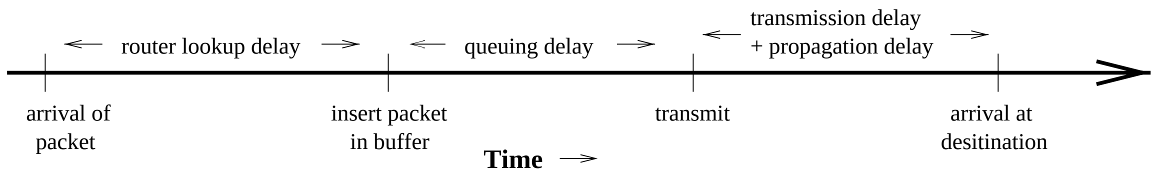

Figure 1: Timeline of Events Used in Buffering Algorithms

that lookups are performed in parallel, this event is still needed to ensure that packets appear in the buffer in the correct order.

Each time an event that writes a packet to the buffer is executed, a buffer element is added to a FIFO queue. This element represents the message in the buffer. Before attempting to place a new element into the buffer, all elements representing packets already sent are removed. At this point the actual amount of space available in the buffer at this time is known. If there is not enough space for the new packet, this packet is dropped. Otherwise a new event is generated to deliver packet to its destination and a new placeholder element added to the FIFO queue.

The nolist_opt buffer object is similar to the standard_opt buffer object, except that it uses a fluid flow to model the draining of packets from the buffer. This eliminates the need for the FIFO queue of placeholder objects at the cost of a small loss of modelling accuracy. In this case rather than knowing the exact time that each packet leaves the buffer, as in the other buffer implementations, a continuous model is used to represent bytes leaving the buffer. In many simulations the small amount of inaccuracy is unlikely to be an issue, though this may not be good for some simulations such as studies of subtle TCP behaviours over congested networks. For that reason, the standard_opt buffer objects are instantiated by default in IP-TN. Users have to explicitly request nolist_opt in the simulation input file if they require extra performance and are prepared to accept the small loss in accuracy.

4.2 Preemptive Buffers

In keeping with the way that preemptive queuing is implemented in Cisco routers, the preemptive buffers in IP-TN/E are implemented as a set of FIFO queues, with preemption occurring by selecting which FIFO to remove the next packet from. A representation of a preemptive queuing buffer is shown in Figure 2. The decision of which FIFO to place a packet in is determined by its DSCP value and a configurable PHB map associated with the buffer. The number and size of FIFO queues employed, and mapping from DSCP value to each of the queues are all configurable using the simulation input file. The user can specify an arbitrary number of queues, though there would be little point in specifying more than 64 since that is the number of possible bit patterns in the DSCP field.

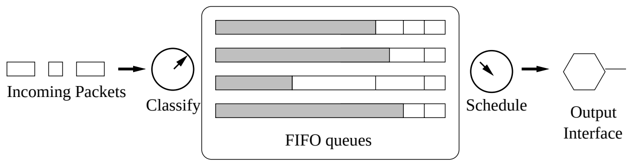

Figure 2: Congestion Management

4.2.1 Priority Queuing

For most experiments TeleSim will perform on SuperNet using priority queuing, four queues will be used since this is the number used by the Cisco Priority Queuing implementation.

For testing purposes two types of priority queue buffer are currently available, *ds_priority* and *ds_priority_opt*. These both provide the same functionality, but keeping both enables validation and makes it possible for the the utility of any performance optimisations performed on the latter to be assessed.

With the ds_priority buffers, when an arrival event occurs a BI event is created and scheduled for the actual buffer insertion time. When the BI event is delivered the DSCP value in the packet is examined and a lookup performed in the PHB table to determine into which queue the packet should be placed. Then a variable is checked to determine if there are packets in any of the FIFO queues. If there are none, this packet can be sent immediately. Otherwise the packet is placed into the appropriate FIFO. If the buffer was empty a CQ event is generated timestamped at the time that the packet has left the buffer. When a CQ event is received, the length of the packet that was dispatched when this event was generated is subtracted from the used byte count for the particular FIFO queue that the packet traversed. Also a check is made to determine if there are any more packets waiting to be sent. If there are, the first packet from the highest priority non-empty FIFO is selected. An arrival event scheduled at the destination host and a new CQ self event are generated.

The ds_priority_opt network buffer is currently quite similar to ds_priority except that for any packet that would be placed in the high priority queue, as long as there is space in the buffer, an arrival event is scheduled immediately for this packet. This means that events representing some packets can be sent to their destination sooner thus providing additional lookahead. Note that CQ self events are still required.

4.2.2 Custom Queuing

As is the case of priority queuing, two custom queuing implementations are currently available which both produce the same simulation results. The *ds_custom* buffer provides a simple implementation while the *ds_custom_opt* type adds a lookahead optimisation.

In the case of custom queuing the modeller specifies the number of bytes that are serviced for each of the FIFO queues. The scheduler for the buffer will continue to send packets from the current queue until the queue is empty or until the number of bytes sent exceeds the allocated service. At this point the scheduler moves on to service the next queue.

When an arrival event occurs, a BI self event is generated. When the BI event arrives the queue that the packet should be mapped to is obtained from the PHB table. If the entire buffer is empty (i.e., there are no packet contained in any of the FIFO queues) the packet is sent immediately and a CQ event generated. Otherwise the packet is placed into the appropriate FIFO, or dropped if there is not enough room in this queue. When a CQ event occurs, the queue scheduler determines which queue it should currently be servicing. If there is a packet in this queue, this packet is sent. Otherwise the queue scheduler moves to the next non-empty FIFO and sends a packet from this queue.

The operation of ds_custom_opt buffers differs from that of ds_custom buffers in that when an a BI event occurs, if the queue scheduler is currently working on the queue the packet should be inserted into, the packet can be sent immediately. This provides additional lookahead.

5 PERFORMANCE RESULTS

This section provides performance results for IP-TN simulations using different buffer implementations. It starts by comparing the performance of the four buffer types: standard_opt, nolist_opt, ds_priority_opt and ds_custom_opt to the performance of simple_no_opt. Then results for experiments that compare the performance of the optimised DiffServ buffers to the unoptimised versions for traffic with various mixtures of DSCP values are presented.

The results of experiments show that many factors effect the performance of simulations using different buffer implementations. One factor is the number of events required to model the passage of a packet from the point that it arrives at a router to the time that the packet arrives at the next host or router. Other involves lookahead which is the amount of time into the future that the occurrence of events at an LP can be predicted. One form of lookahead in a network model is the propagation delay of a link; i.e., how long it takes for one byte of data to pass along the link. Lookahead optimisations attempt to dynamically discover more lookahead during runtime. The buffers of types standard_opt, nolist_opt, ds_priority_opt and ds_custom_opt all use dynamic lookahead optimisation to send the packet arrival event to the next destination host as soon as possible, thus increasing the destination host's knowledge of the future thus possibly increasing the size of its current, or next execution window.

To take any advantage of dynamic lookahead in the simulation using priority queuing buffers, there must be packets logically in the highest priority queue. These packets may only be logically there since in the optimised implementation these packets can be sent to their destination host at the point when they are written into the network buffer. In the custom queuing case, there must be packets logically in the buffer's current service window.

It should be noted that simply comparing the runtime performance of buffers implementing different strategies fails in many cases. In particular if one buffering strategy leads to more packets being dropped than another strategy, then the overall number of events in the simulation run will be different. By dropping a packet on one link the events that would have carried the packet on to its destination are not required. In the case of closed loop traffic stream such as TCP, extra events may be required to carry packets that are resent.

The amount of advantage gained by using a dynamic lookahead optimisation depends on the amount of static lookahead available. For example if there is a large propagation delay on the network link the advantage gained by the extra lookahead from the optimisation is small.

The results presented in this section use the network model shown in Figure 3. This consists of two LANs each with four hosts connected to a router using 10Mb/s Ethernet. The two routers are connected by a 10Mb/s link. Although this is network is relatively small, it can be used to demonstrate the performance properties of the various buffer implementations. For all the experiments, all network buffers are of type standard_opt apart from the buffers on the link between the two routers. Traffic is routed between

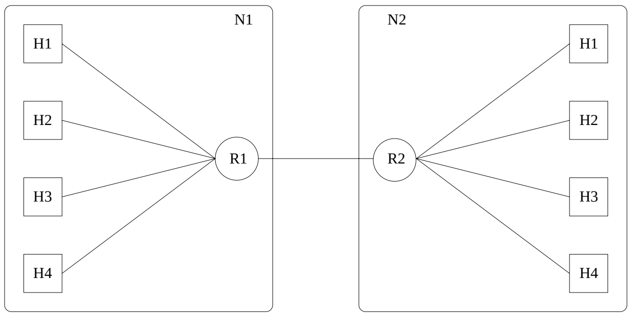

Figure 3: Network Model

pairs of hosts, with traffic on host H_i on LAN N_1 being sent to host H_i on LAN N_2 and vice versa. All the tests for which results are presented were performed using two processors of a Compaq AlphaServer ES40 with 833MHz Alpha Ev68 processors, 4GB of RAM, running the Tru64 Unix operating system.

Figure 4 compares the performance of simulations using different network buffer types on the router interfaces. As already stated it is a little dangerous to compare the performance of buffers implementing different policies since these policies have additional performance related effects on the simulation run. However, these results show interesting properties of the various implementations. These results should be considered preliminary in the sense that testing of many more simulation models are required to fully explore the performance implications of using buffers implementing different policies and different optimisations.

The results in Figure 4 are normalised to the performance of simulations where the simple_no_opt buffers were employed on the router links. Experiments were performed with the propagation delay between the two routers set to values between 10E-7 and 10E-1 seconds. For this set of experiments the performance of standard_opt and nolist_opt were similar showing that the amount of work done calculating the buffer usage for nolist_opt was similar to the work done maintaining the placeholder list in standard_opt. The performance of nolist_opt has been seen to be better in other simulation studies. The fact that the simulation model used required considerably less than the 8MB of cache available on each of the Ev68 processors may have resulted in the memory operations required for standard_opt buffers being very inexpensive. Further experimentation with larger models will be required to fully understand this.

There is an obvious trend that the larger the static lookahead, i.e., the larger the propagation delay between routers, the less is gained from using a dynamic lookahead optimisation. With a propagation delay of 10E-5 seconds the optimised buffer versions are running over 2.7 times faster than the unoptimised versions. With a propagation delay of 10E-1 this advantage is down to 1.2 times faster.

In this set of experiments, the use of the non-optimised buffers resulted in the same runtime performance. The use of optimised preemptive buffers resulted in lower performance than achieved using the optimised non-preemptive buffers. Given the traffic in this set of experiments that was mainly, perhaps unrealistically marked using the EF DSCP, the ds_priority_opt outperformed ds_custom_opt. The results that follow using packets marked with different DSCP values provides a more thorough comparison of the performance obtainable using these DiffServ enabled buffers.

The relative performance of ds_priority_opt buffers and ds_priority buffers is shown in Table 2. The top two rows are for experiments where traffic is flowing in one direction. The second two rows present results where traffic flowing in both directions. Results for experiments with inter-router propagation delays between 1E-7 seconds and 1E-3 seconds are presented. The five rows headed, T1 through T5 are for traffic labelled with different DSCP values as shown in Table 3. Each traffic stream from H_i on N1 to H_i on N2 used the same DSCP value for all its packets.

Table 2: Relative Performance for Priority Queuing

Traffic	Delay	T1	T2	T3	T4	T5
1-way	1E-3	1.09	0.99	0.99	0.99	0.99
	1E-7	2.40	1.40	1.18	1.02	1.00
2-way	1E-3	1.07	0.92	0.93	0.97	0.97
	1E-7	1.38	1.14	1.05	0.99	0.99

Table 3: DSCP values for Priority Queuing Tests

Host	T1	T2	T3	T4	T5
H1	EF	EF	EF	EF	SBE
H2	EF	EF	EF	SBE	SBE
H3	EF	EF	SBE	SBE	SBE
H4	EF	SBE	SBE	SBE	SBE

The relative performance of ds_custom_opt buffers and ds_custom buffers is shown in Table 4. This table is laid out in a similar way as Table 2. The DSCP values for traffic used in these experiments is shown in Table 5.

Table 4: Relative Performance for Custom Queuing

Traffic	Delay	T1	T2	T3	T4	T5
1-way	1E-3	1.01	1.03	1.05	1.01	1.04
	1E-7	1.60	1.38	1.72	1.22	1.69
2-way	1E-3	1.02	0.99	0.98	0.99	1.03
	1E-7	1.25	1.15	1.25	1.06	1.26

Table 5: DSCP values for Custom Queuing Tests

Host	T1	T2	T3	T4	T5
H1	EF	EF	EF	EF	EF
H2	EF	EF	EF	AF41	AF41
H3	EF	AF41	AF41	AF41	AF31
H4	EF	AF41	AF31	AF31	AF21

For Priority Queuing the lookahead optimisation works when there is enough traffic mapped to the highest priority queue. For Custom Queuing, the lookahead optimisation is possible when traffic arrives in the service window of the queue currently being serviced. If only one queue is being serviced (i.e., all DSCPs mapped to one queue) then the lookahead optimisation can be beneficial whenever there are packets buffered. With an equal amount of each DSCP marked traffic performance is also good since there is a good chance of being able to take advantage of the lookahead optimisation for each queue serviced. For cases where the traffic is mapped to different buffers the performance is less

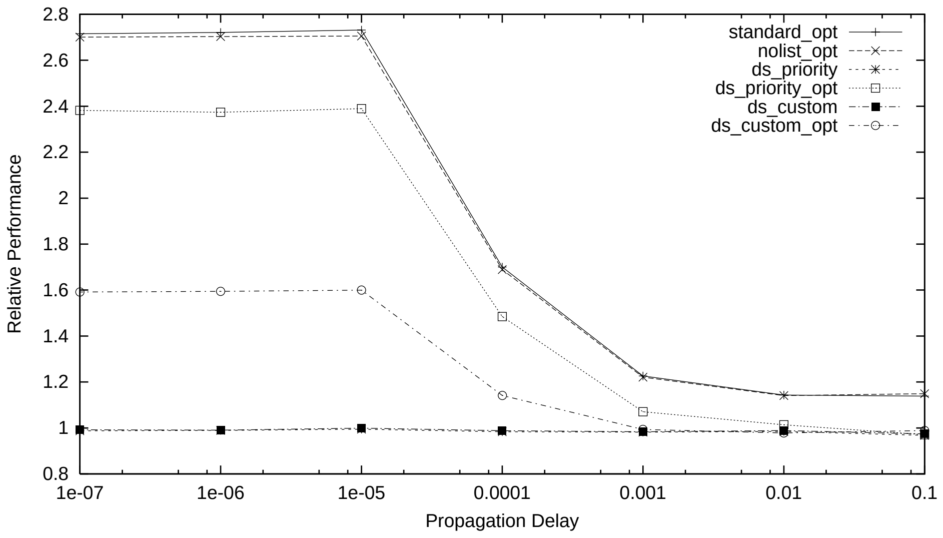

Figure 4: Performance of Different Network Buffers Relative to that Achieved with simple_no_opt Buffers

good due to the having some FIFOs for which there is little opportunity to take advantage of the optimisation.

6 SUMMARY

This paper described issues encountered adding DiffServ to an IP level network simulator using a channel based parallel discrete event simulation kernel. In particular it showed some of the difficulties in implementing the preemptive network buffers required by DiffServ. Optimisations that reduce the number of events required to simulate the movement of packets through non-preemptive buffers are difficult, and sometimes impossible for preemptive buffers. Optimisations that improve the level of dynamic lookahead for non-preemptive network buffers are more difficult to implement and not able to expose as much lookahead in most cases.

Results were presented showing the different performance levels that can be achieved for simulations using different types of non-preemptive and preemptive network buffers. Experiments show that greater levels of static lookahead not only limit the benefit achieved using dynamic lookahead, but in some cases cause buffers implementing dynamic lookahead optimisation to perform worse than simpler buffer implementations. For cases where dynamic lookahead was beneficial, performance advantages of up to 2.72 times were shown for optimised non-preemptive buffers over simpler non-preemptive buffers and up to 2.4 times for optimised preemptive buffers over simpler versions implementing the same buffer management strategy.

REFERENCES

Blake, S., D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. 1998. RFC 2475 - An Architecture for Differentiated Services.

Bryant, R. 1977. Simulation of Packet Communication Architecture Computer Systems. Technical Report No. MIT/LCS/TR-188. MIT.

Chandy, K., and J. Misra. 1981. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Communications of the ACM* 24 (11):198-206.

Cisco Systems. 2002. Internet Operating System Release 12.2. Configuration and Command References. Available online via `<http://www.cisco.com/>` [accessed March 29, 2003].

Heinanen, J., F. Baker, W. Weiss, and J. Wroclawski. 1999. RFC 2597 - Assured Forwarding PHB Group.

Jacobsen, V., K. Nichols, and K. Poduri. 1999. RFC 2598 - An Expedited Forwarding PHB.

Kiddle, C., R. Simmonds, C. Williamson, and B. Unger. 2003. Hybrid Packet/Fluid Flow Network Simulation. *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*.

Nichols K., S. Blake, F. Baker, and D. Black. 1998. RFC 2474 - Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.

Nicol, D. 2002. Analysis of Composite Synchronization. *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*.

Nicol, D., and J. Liu. 2001. Composite Synchronization in Parallel Discrete-Event Simulation. *IEEE Transactions on Parallel and Distributed Systems* 13:(433-446).

Simmonds, R., R. Bradford, and B. Unger. 2000. Applying Parallel Discrete Event Simulation to Network Emulation. *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*.

Simmonds, R., C. Kiddle, and B. Unger. 2002. Addressing Blocking and Scalability in Critical Channel Traversing. *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*.

Williamson, C., R. Simmonds, and M. Arlitt. 2002. A Case Study of Web Server Performance Using Parallel WAN Emulation. *Performance Evaluation* 49 (1):111-127.

Williamson, C., and Q. Wu. 2002. A Case for Context-Aware TCP/IP. *ACM Performance Review* 5 (1):11-23.

Xiao, Z., B. Unger, R. Simmonds, and J. Cleary. 1999. Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation. *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*.

AUTHOR BIOGRAPHIES

ROGER CURRY is an M.Sc. student in the Department of Computer Science at the University of Calgary. He was previously employed as a research associate with the TeleSim research group working on extending and maintaining the IP-TN simulation system.

ROB SIMMONDS is an Adjunct Assistant Professor in the Department of Computer Science at the University of Calgary and the Distributed Systems Architect for the WestGrid project. His main research interests are in parallel simulation, parallel network emulation and distributed systems. Dr. Simmonds holds B.Sc. and Ph.D. degrees from the School of Mathematical Sciences at the University of Bath in England.

BRIAN UNGER is the President and CEO of iCORE, the “informatics Circle of Research Excellence”, and is a Professor of Computer Science at the University of Calgary where he leads the TeleSim research group with Dr. Simmonds. iCORE is a not-for-profit corporation aimed at attracting and funding research leaders in the information and communications sciences at Alberta universities. Dr. Unger received a Ph.D. in Information and Computer Science from the University of California, San Diego, M.Sc. from the University of California, and B.Sc. from Loyola University, Los Angeles.