Speedup vs. Simulation Granularity

Jong-Suk Ahn and Peter B. Danzig
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
{jahn,danzig}@usc.edu
213-740-4780 (Office) 213-740-7285 (Fax)

Abstract

This paper describes a packet network simulator whose timing granularity can shift continuously from fine, packet-level detail to coarse, conversation-level detail. Simulation run time decreases with coarser timing granularity, but the details in the underlying model become faded as the timing granularity coarsens. The finer the granularity, the slower but more precise the simulation. If a simulation becomes resource limited, it is possible to coarsen the timing granularity to scale the simulation larger.

This paper introduces a new simulation technique to speedup simulation of high speed, wide area networks. The new technique can yield order of magnitude speedup and memory savings on simulations of large-scale packet networks. The speedup is achieved by introducing a degree of approximation into abstracting packet streams. We call this technique *Flowsim*. Flowsim can yield different simulation metrics than packet simulation due to its different degree of simulation granularity. We have replicated simulations presented in the literature and, in this paper, show that it is frequently possible to employ coarse timing granularity when studying flow and congestion control algorithms.

This paper motivates the need for faster and less memory intensive simulators, introduces two other simulation techniques which can together double the performance of well written simulators, describes Flowsim's traffic representation scheme and internal algorithms, explores the tradeoff between speed and accuracy, and explores some of the limitations of traditional simulation. It demonstrates that speedups of a factor of 50 are possible when simulating ATM networks.

1 Introduction

As network researchers, we evaluate new routing, admission control, congestion control and flow control algorithms by simulation rather than by analysis. We rarely employ analysis because it cannot capture the heart of the algorithms we wish to study. Unfortunately, simulations are time consuming and memory intensive.

The simulation performance depends on the timing granularity of simulations. The finer the granularity, the slower but more precise the simulation. Fine grain, packet-level simulation models every individual packet's behavior. As networks carry more packets, packet-level simulation becomes slower. Simulation run time decreases with coarser timing granularity, but the details in the underlying model become faded as the timing granularity coarsens. In coarse grain, conversation-level simulation, the whole packet stream of each conversation can be represented analytically with a few parameters, for example the average number of packets per second.

This paper describes a packet network simulator whose timing granularity can shift continuously from fine, packet-level detail to coarse, conversation-level detail. The new simulator represents packet streams either in packet-level detail or in conversation-level detail whenever it is appropriate. The new simulation technique can cut the cost to simulate high bandwidth, wide area packet networks by an order of magnitude. This cost reduction comes at the price of introducing an adjustable degree of approximation into abstracting packet streams. The higher the memory savings, the faster the simulator, but the less accurate the computed metrics. We call our technique *Flowsim*, because it deals with packet flows, rather than individual packets.

This paper, through the example below, explains why analytical techniques have failed to model modern packet networks, why packet simulation is computationally costly, and why a simulator's state can grow quickly.

Figure 1 shows a wide-area network topology that flow and congestion control researchers might study. This network's 55 nodes forward the packets of 250 conversations; only 7 conversation endpoints and 32 gateways are indicated. Node labels "S", "D", and "G" indicate traf-

Simulating five minutes of activity on a network the size of today's Internet would require gigabytes of real memory and months of computation on today's 100 MIP uniprocessors. Our long term research goal is to develop the necessary simulation technology to evaluate such large internets.

Section 2, below, describes two techniques applicable to any packet simulator that double performance for FIFO switch scheduling algorithms. Section 3 shows how Flowsim's packet representation scheme can save memory, and Section 4 explains why Flowsim is an order of magnitude faster than packet simulators. Section 5 describes our simulations and interprets their results. We discuss insights into simulating networks and consider some of Flowsim's limitations in Section 6.

2 Two Speedup Techniques

Packet network simulators, like all discrete event simulators, are built around an event list. When a node forwards a packet along a link towards a neighboring node, it actually inserts two events into the event list; one with time set to the service time to put the packet on the wire plus the current time and one with time set to the sum of the link's propagation delay, the service time, and the current time. It includes the service time because few packet switches employ cut-through routing and hence cannot switch a packet until its last bit arrives.

When simulated time catches up with the first event, the node can dispatch a buffered packet upstream. When time catches up with the second event, the destination node retrieves the pointer to the packet header and either delivers the packet to the application layer or enqueues it towards the hop.

2.1 Optimization for FIFO Switches

We have developed a technique that can double the performance of packet simulators when switch scheduling is FIFO. This technique expands the *outbound lookahead* of FIFO switches to be enough large to dispatch all buffered packets at one time. The discrete-event simulation literature describes a node's lookahead as the time beyond its logical clock that a node can simulate without violating causality. A node has two different lookaheads; *inbound* and *outbound lookahead*. A node can buffer packets which will arrive no later than its inbound lookahead and forward packets which will depart no later than its outbound lookahead. The bigger outbound lookahead, the more packets can depart in advance and the less events to inform the time at which a next packet departs. Without outbound lookahead, modeling each packet's departure needs two events as explained above.

Without the new technique, outbound lookahead would be at most the same as inbound lookahead. The outbound lookahead of FIFO switches can be increased to infinite since packet arrival can not force a previously arrived packet to be dropped or affect the time at which it should depart the switch. The technique directly forwards packets upstream without ever serializing them in the buffer since their arrival time and dropout can be determined in advance. Compared to conventional implementations, this halves the number of simulation events necessary to model packet departure to neighboring nodes. Upon receiving a packet arrival event, a node immediately forwards it towards the next node or, if this packet should overflow, discards it immediately. The node sets the arrival event to occur at time according to the sum of its queueing delay, its node service time, and the link propagation delay. All that is necessary is for the node to keep track of its *virtual queue length* as well as the time at which the previous packet departed. For example, if a packet arrives at time 3, but the virtual queue won't clear until time 4, then the packet is enqueued to arrive down stream at time max(3, 4) + packet service time + link propagation delay.

2.2 Reducing Global Event List Length

Even well optimized packet network simulators can spend 25%-50% of their time inserting and deleting events; flow control and packet forwarding are not themselves terribly costly. Event lists are built from priority queues. Although naive priority queues work best with a small number of events, sophisticated ones exhibit O(1) behavior even with a large number of events if the variation in the number of events remains low [3, 22]. This leads us to ponder how the number of events in the event list fluctuates with time.

The bandwidth-delay product of a link establishes the maximum number of packets that it can carry. Summing this maximum over all links establishes the maximum number of events in the event list. If network routing and traffic sources remain fixed throughout the simulation, the variation in the number of packets in the simulator depends on the traffic characteristics and oscillates with the flow control window sizes. The faster and longer the link, the higher the variation of the number of events in the simulation. This variation, which is not a problem when simulating a 1.544 Mb/s T1 link carrying IP packets, seriously degrades the event list performance when simulating gigabit links carrying ATM cells.

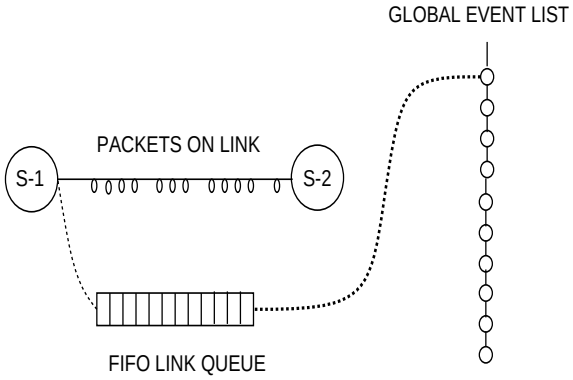


Figure 2: Reducing variation in the event list's length.

Flowsim reduces the event list's length and variance in length by inserting into the event list only the next-to-arrive packet from each link. The events for all other in-transit packets remain in the appropriate link's *event FIFO*. The simulator, upon deleting the next-to-arrive packet from the head of the event list, moves the head of the appropriate link's FIFO into the event list. Figure 2 illustrates a link, the link's event FIFO, and the (global) event list. The heavy dotted line represents the relationship between the next-to-arrive packet in the event list, and the subsequent packet in the link's event FIFO. Keeping all but the leading packet in the FIFO rather than directly in the global event list can reduce global event list processing time by 50%.

2.3 Adjustable Calendar Queue

This section addresses a problem with *calendar queues*, a sophisticated priority queue used as the global event queue in our simulator. Table 1 lists random seed values for five simulation runs and the corresponding execution times. The second row shows, surprisingly, that a change on the random seed causes one simulation run to take twelve times more than other runs. It is because calendar queues do not adapt their structure dynamically to the event distribution changing over time.

Calendar queues take after calendars on which people write down their to-do-lists. A calendar queue is an array of pointers called buckets. Each bucket has a width representing a certain amount of time and enqueues events whose time falls within the bucket's width in order of their timestamps. The calendar queue performance depends on how evenly events are distributed over buckets. With too big a width, a long line of events forms at only a few buckets. On the other hand, too small a width separates adjacent events across lots of buckets so that a dequeue operation needs to visit a lot of buckets before reaching the next event.

Random Seed	Exe. Time(Sec)
17	2,166
25	28,022
52	1,877
78	1,921
98	1,969

Table 1: Execution time of a packet simulator while varying random seed

The huge execution time of the second row in Table 1 results from continuous use of too small bucket width computed at one time. Calendar queues calculate the bucket width by averaging some number of inter-event gaps only when the number of enqueued events doubles or halves comparing to the number at the last time when the bucket width is computed. When the number of enequeued events does not vary, calendar queues will use the same bucket width during the rest of simulation duration. Let's assume that when a calendar queue determines the bucket width, the calendar queue represents time interval [3, 10] which can be divided into two sub-intervals according to event distribution; [3, 4] with 0.1 millisecond average inter-event gap and [4, 10] with 1 millisecond average inter-event gap. The bucket width determined based on [3, 4] would be too small for events in [4, 10]. We plan to implement a mechanism in calendar queues to monitor the event distribution and change the width dynamically based on on the event distribution, not on the number of events.

3 Representation

Our third optimization technique represents an entire group of closely spaced packets as a single event, enabling Flowsim to represent wide-area network traffic with fewer events than a tradititional simulator. Below, we show how Flowsim represents packets in switches and links.

3.1 Reducing Memory Usage

Because an event can represent many packets, Flowsim's event structure includes much of the protocol header information present on real networks, plus additional fields for timing and event management. The exact structure of an event depends on whether or not the simulator simulates a particular protocol processing stack, or whether some of this detail has been abstracted. The simulator out of which we developed Flowsim represents

Message Fields
`packet *next`
`int packet_type`
`short Source_Node`
`short Destination_Node`
`short packet_size`
`time transmission_time`
`long sequence_number`
...
Event Fields
`time event_time`
`void (*evt_handler)()`
`Event *next`
`Node_ID Node`
...

Figure 3: Typical fields in an event.

the TCP/IP protocol stack in fine detail. Figure 3 details some essential fields.

It's relatively easy to reduce memory consumption per event by factoring out the event's *Message fields*. One can replace both `Source_Node` and `Destination_Node` with a pointer to a `Conversation_Descriptor` shared by all a conversation's packets. Even the `packet_size` can be factored into the `Conversation_Descriptor` if, like FTP or audio, the conversation uses a single packet size all the time. Beyond such factoring, further reducing memory consumption requires making approximations.

3.2 Flowsim on Links

Jain noted that back-to-back packets on local area networks tend to arrive from the same source and are headed to the same destination [18]. He called such a burst a *packet train*. He said that if the spacing between two packets exceeds some *inter-train gap*, that these packets belong to separate *trains*. In subsequent years, several protocol stack implementations were optimized to exploit this phenomenon [4, 16]. More recently, other investigators found that due to the high degree of multiplexing, back-to-back packet trains are not as prominent on wide-area networks as they are for local area networks [14, 10]. Researchers are already calling for a definition of packet trains for wide-area-networks [21].

We now describe how, motivated by wide-area-network packet trains, Flowsim represents closely spaced packets. Each link in Flowsim places a conversation's packets on a linked list of packet trains called a *flow descriptor*. Each conversation traversing a link has its own flow descriptor. When a node originates or forwards a packet onto a link, the simulator appends it to the corresponding flow descriptor.

Train Descriptor Fields
`int packet_count`
`time lead_time`
`time tail_time`
`time link_idle_time`
`long sequence_number`

Figure 4: Fields of a train descriptor.

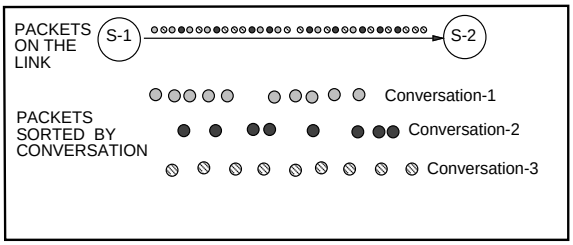
Each packet train in a flow descriptor contains the information listed in Figure 4. A flow descriptor appends a new packet to an existing train if some other packet was also added to this train no longer than an inter-train gap ago. If the flow descriptor appends the packet to the train, it increments the train's `packet_count` and extends its `tail_time` accordingly. If it cannot append the packet to the train, it creates a new train for the packet and chains it to the other trains it carries. Flow descriptors also create new trains for protocol specific reasons. For example, if packets carry sequence numbers, then a packet with a non-sequential sequence number starts a new train.

The upper half of Figure 5 illustrates a physical link carrying packets from three conversations. The lower half of the figure illustrates Flowsim's internal representation of the same. Each train's `packet_count` is shown, as well as each train's first and last packet. Line segments represent packet trains, and chained packet trains belong to the same flow descriptor and hence conversation.

Packet trains save memory because they represent many packets in the same space of one. However, since Flowsim eliminates exact packet arrival times, it can only approximate the location of packets in trains.

3.3 Event List and Flow Descriptors

We described in Section 2.2 how we reduced the event list's length by only inserting each link's lead packet. Recall that upon dispatching the event corresponding to a link's lead packet, we insert the link's next packet into the event list. We could have inserted the lead packet from each flow descriptor into the event list, and upon dispatching this event, approximated an arrival time of the flow descriptor's next packet and inserted this next packet into the event list. The disadvantage of this solution is that the global event list would have grown by the number of conversations traversing each link. To



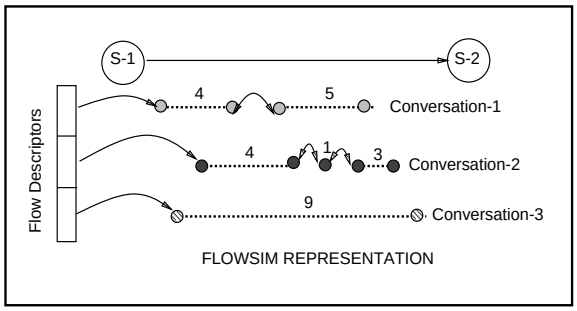
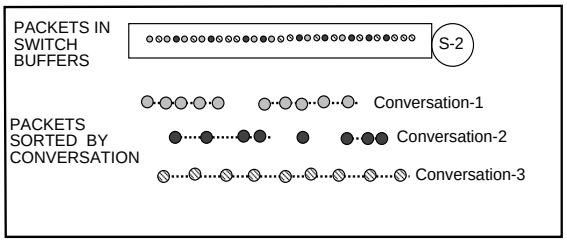


Figure 5: Flowsim's packet representation.



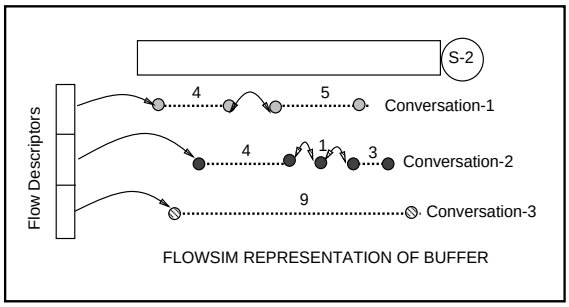


Figure 6: Flowsim at switches.

avoid this, we maintain a separate priority queue for each link's flows and place only each link's lead packet into the global event list.

Many protocols generate nearly as many acknowledgements as data packets. For example, most TCP implementations acknowledge every or every other data packet. Although acknowledgement packets tend to be smaller than data packets, simulators represent them both with the same size events. Because half the possible memory savings may be in the form of acknowledgements, Flowsim places acknowledgement packets into trains, just as it does data packets.

Notice that trains are created dynamically. Traffic sources emit packets which the simulator places into trains. The representation saves memory at the cost of introducing variations in timing into the simulation. Notice that decreasing the inter-train gap recovers exact packet-by-packet simulation. While debugging Flowsim, we frequently set the gap to zero to see that Flowsim and our packet-by-packet simulators agreed.

3.4 Flowsim at Buffers

We use flow descriptors to hold packets enqueued at network switches, and, as the next section describes, use these flow descriptors to move entire packet trains through network switches simultaneously.

Figure 6 illustrates how Flowsim represents a switch buffer. Packets that arrive at a switch get inserted into flow descriptors, just as if they were being placed on a link. FIFO switches always read from the train with the earliest `lead_time`. Each time the switch reads from a train, that train's new `lead_time` is approximated or, if the train's packet count reaches zero, it is discarded. A priority queue per switch buffer identifies the earliest train. Note that a train's `lead_time` is the approximate time at which the train's leading packet is enqueued at the switch. Switches with other scheduling disciplines must employ a priority queue that orders their flow descriptors as needed.

If a switch discards a packet for lack of buffer space, the flow descriptor insertion algorithm detects a gap in sequence numbers and automatically causes the packet train to stop growing and starts a new train.

3.5 Distribution of Packets in Trains

Flowsim approximates that packets within a train are constantly spaced along the train. Flowsim does not approximate that packets are distributed more randomly along the train, because it would complicate the hybrid algorithm described later. Flowsim does not, as implied above, extract individual packets off of flow descriptors and switch them individually. Instead, as described in the next section, Flowsim brings entire trains of packets through network switches in a single, hybrid operations. However, it is possible to use just Flowsim's packet representation to save memory and not use its hybrid technique to save simulation time.

When not using hybrid simulation, Flowsim regenerates individual packets. Upon delivering a train's leading packet, Flowsim must select the time for the train's new leading packet and adjust the train's `lead_time`.

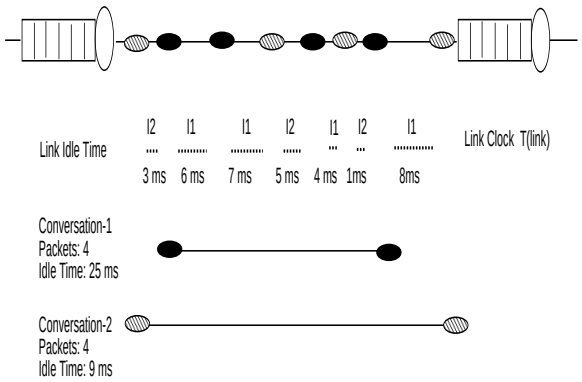


Figure 7: Regeneration algorithm uses the link idle time recorded for each train.

The regeneration algorithm places an equal slice of the link idle time between a train's packets by adding this slice to the packet's service time. Recall that switch service time is the packet size divided by the outgoing link bandwidth. Extending the service time maintains spacing between packets. All of our simulations employ a traffic type dependent, constant packet size.

Figure 7 illustrates two conversations that fall in two packet trains. Below the figure of the packets and queues is a sequence of line segments that indicate the amount of idle time that precedes each packet. The bottom of the figure shows each train's duration, the number of packets in each train, and sum of the idle time before the train's packets.

4 Hybrid Simulation

Flowsim saves CPU usage over packet-by-packet simulation by simultaneously conducting entire packet trains through network switches. It does this by exploiting two ideas from the field of conservative, parallel distributed simulation [23]. On the example topologies we tested, this feature makes Flowsim 3 to 11 times faster than the optimized packet-by-packet simulator from which it was developed. Obviously, like the flow descriptor and train representations, this algorithm trades accuracy for efficiency.

4.1 Conducting Trains through Switches

Conducting entire trains through switches would be simple if each switch could independently unlink the train with the earliest **lead_time** from its link buffer and deposit the train, appropriately modified, on the next hop's link buffer. Unfortunately, two features of packet trains prevent switches from doing this. First, because trains overlap each other, the first train to arrive at a switch may not be the first train to depart. This means that to conduct the first train, all trains that overlap with it must be dealt with at the same time. Second, each of Flowsim's switches maintains its own *logical clock*. A switch cannot process trains beyond the earliest time that, on any of its links, a new train might arrive. Hence, not only must Flowsim's switches process trains simultaneously, but they must obey the causality constraints of conservative, distributed, discrete-event simulation.

```
(1) Calculate Lookahead L_i of node

(2) While (train = GetEvent()) {
    (3)    If (train->lead_time > L_i) {
    (4)       InsertEvent(train);
              break;
    (5)    } else if (train->tail_time > L_i)
    (6)       SplitEvent(train);
    (7)    Consume(train);
}
(8) Assign buffer overflow losses;
(9) Calculate train departure times;
(10) Advance local clock to L_i;
(11) Forward trains that depart before L_i.
     Delay remaining trains in output buffer.
```

Figure 8: Flowsim's train processing algorithm.

The discrete-event simulation literature describes a node's *lookahead* as the time beyond its logical clock that a node can simulate without violating causality. For packet networks, a switch's lookahead increases linearly with the propagation delay of its links. The higher the propagation delay, the greater the lookahead. Since Flowsim's performance increases with lookahead, its performance gets better as the bandwidth-delay product increases.

4.2 The Switch Algorithm

This section walks through the algorithm sketched in Figure 8. Flowsim's global scheduler invokes this algorithm on the node with the oldest ready-to-depart (earliest **lead_time**) train in the system. Figure 8 denotes this as node i. In step (1), node i calculates its lookahead – the minimum amount of time through which it can process trains without violating causality. Once it knows its lookahead, it can fetch all the trains from its incoming links with **lead_time** less than its lookahead.

Node i's lookahead, L_i, is the minimum of the lookahead along its incoming links. The lookahead of an incoming link from node j, denoted L_{ji}, is the maximum of (a) the **tail_time** of any incoming train on the link (T_{ji}), and (b) the sum of j's logical clock and the link's propagation delay ($C_j + D_{ji}$).

$$L_i = \underset{over\ j}{Minimize}\ L_{ji}$$

$$where$$

$$L_{ji} = Max\,(T_{ji},\ C_j + D_{ji})$$

If node j is idle, then Flowsim substitutes the logical time at which node j will next be scheduled in place of C_j above. In summary, in step 1, node i predicts the earliest possible arrival time of any train that any of its neighbors can send.

In steps (2)-(7) the node collects trains to process simultaneously. It stops collecting trains when no more trains are ready (step 2) or when it encounters a train that arrives beyond the lookahead time (step 3). In the latter case, it puts this train back in the event list (step 4). In step 5 it identifies trains that straddle the lookahead time and puts the latter half of such trains back on the event list (step 6).

Step 7, invoked each time through the loop, conducts a train into the switch's output buffer, although this is only a temporary step. After collecting all trains in the output buffer, it calculates and assigns packet losses in step 8 and splits the affected trains. It calculates train departure times in step 9, as detailed below. The node advances its clock to the lookahead time L_i (step 10), splits outgoing trains that straddle L_i, enqueues the departed portion of these trains on the appropriate upstream links (step 11), and leaves the undeparted portion of these trains in its output buffers. Note that a split train may be merged together when the node later enqueues it on the flow descriptor. As an optimization, FIFO switches can forward all trains in step 11, even trains that depart after L_i. For priority scheduling disciplines, causality prevents these trains from leaving the output buffers early.

4.3 Approximating Packet Loss

In mapping trains through a switch, buffers might overflow. In step (8), Flowsim's switches calculate packet losses and assign them to trains. It must do this without regenerating individual packets from trains to retain the speed advantage of the train representation. It employs a simple fluid approximation to packet loss. At instants where a new train appears or an old one ends, Flowsim subtracts the number of packets that have departed from the number that have arrived at the buffer, and calculates the number of packets, if any, lost to overflow. If overflows occur during one of these intervals, Flowsim attempts to distribute packet losses to the various trains, according to their packet density.

While this computation is costly, in a large network, most of the switches are not congested. This means that flowsim only infrequently has to work harder, and on average, it gains speedup.

4.4 Approximating Departure Times

After a node splits its trains due to packet losses, it must approximate the time at which each train's leading and trailing packet departs. The departure time of any packet is, of course, the sum of its arrival time at the buffer, the time it waits in the buffer, and the service time to place it on the wire. We consider FIFO switches and more general policies separately.

With FIFO scheduling, we estimate a packet's queueing delay as an average packet service time multiplied by the number of packets that are ahead of it. Calculating the **lead_time** and **tail_time** of a train reduces to counting the number of packets that get transmitted before the first and before the last packet of a train. The number of packets transmitted before a particular packet is the sum of two quantities: the packets in trains that depart before this packet and the fair fraction of the packets of overlapping trains. Consider, for instance, two overlapping trains on the same output buffer. Let one have 10 packets on the interval [2, 8] and the other 7 packets on the interval [5, 12]. Flowsim would approximate that (3/6)*10 of the first train's packets are sent before the first packet of the second train.

With priority scheduling, switches can only forward trains that are ready to depart before the lookahead time. Causality prevents them from forwarding trains that depart after the lookahead time because a higher priority trains could preempt their position in the output buffer. In contrast to FIFO switches which need only order newly arrived trains, priority switches need to re-rank all the trains in the link buffer according to their priority and then calculate loss and departure times. After reordering the trains, priority switches apply the same calculation as FIFO switches to determine departure times.

4.5 Implementation Status

This concludes our description of Flowsim's packet representation and internal algorthms. Flowsim is implemented in Jacobson's and McCanne's *tcpsim*, a simulator which evolved from Berkeley's *REAL* and Columbia's

NEST [11]. In the next section we report Flowsim's speedups and contrast its simulated metrics against our packet-by-packet version of this simulator. For reference, our packet simulator processes 28,400 events per second on a SPARCstation-20 machine when simulating the topology shown in Figure 1.

5 Experiments

We conducted many experiments to evaluate Flowsim's speedup and the tradeoff, in its hybrid mode, between speedup and accuracy. Here, we present the results of simulating large window TCP flow control on two network topologies; one big topology shown in Figure 1 where 55 gateways forward packets of 250 conversations and one small ATM network where 30 traffic sources send 53-byte cells to one destination via a gateway. A practical constraint prevented us from simulating a larger topology: without Flowsim, our traditional packet simulator takes 19 hours to simulate 3 minutes of the big gigabit network. Because three minutes of network time is enough time for four thousand round trip times through the network, we felt that 3 minutes was enough time to see any interesting behavior that might develop.

Our experiments investigated the *stable* unfairness between similarly routed bulk transfer conversations that Floyd identified and termed *phase effects* [13]. Our experiments investigated whether phase effects cause unfairness even with large flow control windows, high network bandwidths, and higher degrees of traffic multiplexing.

5.1 Speedups and Accuracy

The total number of events generated during a simulation determines a simulation's run time. In Flowsim's hybrid mode, longer trains mean fewer events and faster simulations, because Flowsim can process an n packet train in the cost of a few packets. Train length depends on lookahead which is a function of network topology, bandwidth, and link length. Larger lookaheads permit fewer train splittings and increase Flowsim's performance. For these reasons, Flowsim performs well in networks with high bandwidth, long links, small packets and bursty sources.

The longer we stretch trains by increasing the permissible inter-train gaps, the less accurate are simulation's metrics. Figure 9 shows Flowsim's speedup over our traditional packet simulation as a function of inter-train gap. The two upper lines occur for the simulation of the small ATM network. For ATM network simulations, we added an ATM adaptation layer, assembling and disassembling IP packets into 53-byte cells below IP. For the lower line, we physically shortened the bottleneck link between the switch and the destination and lowered its bandwidth to shorten packet train lengths and reduce node lookahead time. In this more congested network, more packet losses occur, more of the bulk transport connections are inactive, and trains tend to be shorter.

For both ATM topologies, Flowsim gets faster as trains grow. Even at zero inter-train gap, when the hybrid algorithm is disabled and Flowsim simulates one packet at a time, the two techniques explained in Section 2 make Flowsim 1.8 times faster than its packet simulator. With 0.05 millisecond inter-train gap, the speedup jumps to 28 and 44 times since after this gap trains begin to convey more than one IP packet. Note that acks come back 0.025 millisecond apart during the congestion period since the 200 Mbps bottleneck link takes 0.025 millisecond to transmit a 512-byte packet. Finally, Flowsim approaches to 56 and 34 times speedup. The lowest line of Figure 9 plots Flowsim's speedup when simulating the big IP network in Figure 1. Note that the bandwidths of Figure 1's central links are lower than the edges, producing lots of congestion. For reference, our packet simulator required 19 hours to simulate the congested Figure 1 topology. As trains stretch longer, Flowsim can achieve 8 times speedup in this topology.

For Flowsim's accuracy, we measured three simulation metrics in the IP network of Figure 1: each conversation's throughput, number of dropped packets, and its one-way delay. For audio traffic conversations, which do not employ flow control, the essential metrics are one-way delay and drop rate. For bulk transfer traffic, the essential metrics are throughput and loss rate.

We first consider the average throughput and the variance between the throughputs of all 78 FTP conversations. For the average throughput of packet simulation, we ran simulations with 8 different random seeds. Figure 10 plots the ratio of Flowsim's results to packet simulation's results. It shows that Flowsim is different than packet simulation by less than 2 % in terms of the average throughput and variance.

Figure 11 shows the packet loss ratio of Flowsim to packet simulation. Two flat lines in Figure 11 are the upper and lower bound of 90 % confidence interval of packet simulation. The two flat lines show a range of how much packet loss can be fluctuated even in packet simulation when the random seed is varied. In this figure, as trains stretch Flowsim tends to drop more packets, up to 15 % with 1 millisecond inter-train gap. It is due to the Flowsim's approximation that packets are evenly distributed in their trains. The Flowsim's approximation can underestimate the traffic burstiness,

open TCP congestion windows wider, and finally lead to more drop.

Figure 12 plots the average throughput ratio of Flowsim to packet simulation over twenty bulk transfer conversations. Figure 12 draws two vertical lines representing 90 % confidence interval over each conversation identifier(cid); one vertical line with circles ends for packet simulation and another with plus ends for Flowsim. Each vertical line shows the range of average throughput that each FTP can achieve during 3-minute simulation time. From Figure 12, we can see that cid 25 sends 25 % more packets in Flowsim than in packet simulation. Flowsim's approximation causes less traffic burstiness, less packet loss and delivers more packets for this FTP conversation. On the other hand, cid 20 sends less packets in Flowsim. It is because the increased throughputs of neighboring FTP's cause more congestion for this FTP. Since most of Flowsim's confidence intervals overlap with the corresponding packet simulation confidence intervals, however we conclude that Flowsim produces similar throughput metrics to packet simulation.

Figure 13 and 14 plot the one-way delay distribution of an audio conversation sending packets from S-1 to D-1 in Figure 1. The minimum one-way delay without queueing delay between the two nodes is 20 milliseconds. In Figure 14, packet simulation predicts the same delay distribution regardless of random seed values. The curves in Figure 13 are distinguishable from packet simulation; Flowsim appears not to affect real-time conversation delay because any decrease in throughput of one conversation is compensated by another conversation's increase. One difference is that as trains stretch longer Flowsim tends to deliver more packets with the minimum delay, 20 milliseconds. With 0.3 millisecond intertrain gap, Flowsim delivers 7 % more packets by 20 milliseconds delay than 0.05 millisecond intertrain gap.

A network algorithm can be more sensitive to the Flowsim's approximation than other network algorithms. In FIFO switches, Flowsim represents packet streams differently, producing different packet loss, driving TCP in a different direction and finally yielding different simulation metrics. For example, when an ON-OFF source generates four packets every 0.1 millisecond, Flowsim delivers one packet every 0.025 millisecond at down stream switches, resulting in different packet loss. Our experiments, however showed that in large-scale networks where packets from a number of conversations get multiplexed, Flowsim can predict the same simulation metrics even with large inter-train gaps.

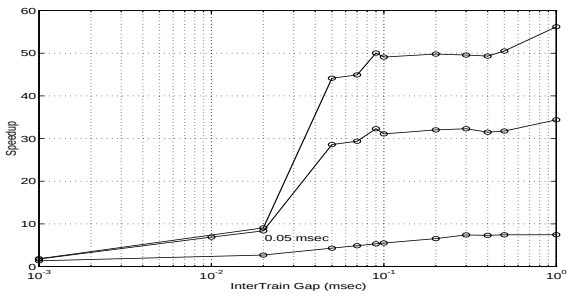


Figure 9: Flowsim speedup over packet simulation

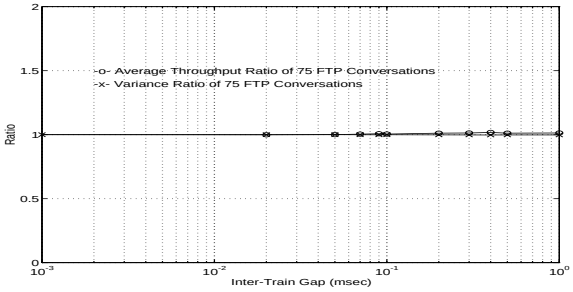


Figure 10: Average throughput and variance

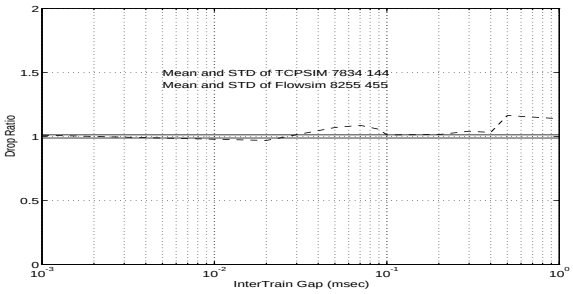


Figure 11: Ratio of Flowsim's to packet simulations packet drops

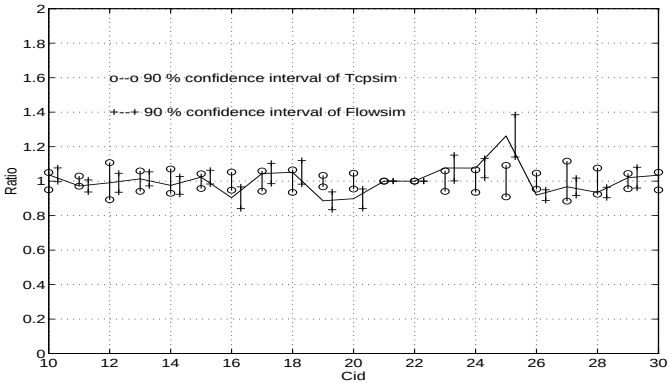


Figure 12: Throughput plot of twenty FTP conversations

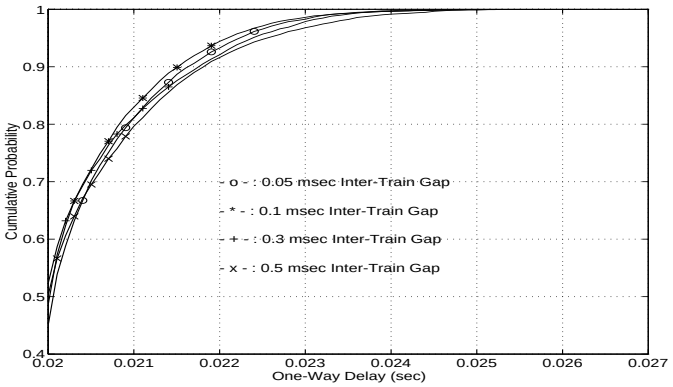


Figure 13: Flowsim's one-way delay distribution

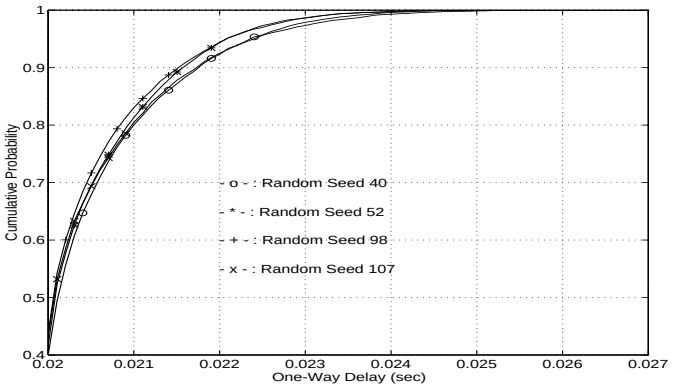


Figure 14: Packet simulation's one-way delay distribution

6 Conclusions

In high bandwidth-delay networks, thousands of packets can simultaneously traverse a given link. This paper investigated a new technique to reduce the cost of simulating such networks. The new technique can reduce simulator memory consumption and processing time by a factor of 3 to 11, although it adds a degree of approximation to the simulation. In addition, this paper introduces two new simulation techniqeus; the first technique reduces the cost of managing the global event list. The second technique halves the number of events necessary to simulate the same network. When simulating ATM networks and when simulating IP over ATM networks, the three techniques, togetger can achieves massive speedups.

Below we review the lessons we learned while evaluating the degree of approximation that Flowsim introduces, speculate on problems to which Flowsim can be applied, and summarize our next steps.

6.1 Notes on Network Simulation

Upon implementing Flowsim, we thought we could quickly evaluate how it affects the simulation of trivial network topologies. This was not the case. Instead, it took over a year of experimentation for us to learn the following lesson: It is not easy to evaluate simulations of small topologies with relatively static traffic sources because the metrics that they yield can be sensitive to minuscule changes in network bandwidths and link lengths [13].

Fundamentally, the question is whether we have a firm enough grasp on the factors that affect a simulation's metrics to evaluate flow and congestion control algorithms via simulation. The lesson is that you cannot learn from your experiments if you misattribute the reason why the metrics differ. You must observe the range of these measures in an ensemble of similar topologies. Future experimenters should report some of their results in scatter plots like our Figure 12. The larger the topology, the more dynamic the traffic models, the higher the degree of traffic multiplexing, the safer the simulation is to interpret.

A simulation run is nothing more than a sample path in a very large event space. Floyd showed that, at least for slow-start TCP, this sample path depends on measures as curious as exact cable lengths. In the final analysis, let the experimenter beware. We find that after a year of experience with Flowsim, that we are confident that Flowsim can be applied to study interesting problems in adaptive routing, flow control, and congestion control. Flowsim also could be used to study call routing and call admission control algorithms in a mix of

real-time and data traffic.

6.2 Extensibility

We applied Flowsim to a study of a network congestion detection algorithm, Tri-S, a variation of which is employed in Vegas TCP [24, 2]. Tri-S monitors the network traffic and stops increasing congestion window before packet loss when the measured traffic congestion is beyond a certain threshold. Figure 15 and Figure 16 plot the congestion window of an FTP competing for a gateway's finite buffer with another FTP. Each figure has two lines; the solid line for a packet simulation and the broken line for Flowsim. In Figure 15, Flowsim shows the same oscillatory behavior of TCP as packet simulation. The two lines in Figure 16 indicate that Tri-S windows approach to a steady state value even though Flowsim predicts a larger window size.

Another important measure of Flowsim's usefulness is whether it can model relevant new network scheduling mechanisms. In addition to FIFO router scheduling, we regularly apply Flowsim to Fair Queueing routers [8]. We have even created a compacted buffer space representation for it. Flowsim and packet simulation agree more closely for Fair Queueing routers than they do for FIFO routers because TCP conversations crossing Fair Queueing switches do not suffer phase effects.

We have sketched but not implemented a way to approximate FIFO+, a scheduling technique designed for multimedia traffic [7]. In FIFO+, the header of each packet carries a field indicating whether the packet is behind schedule or ahead of it. In a simulator, this field can be derived from a time stamp that indicates the exact time at which a source emits a packet. Although Flowsim discards per packet information, it can approximate this timestamp at the cost of two additional timestamp fields per train.

However Flowsim does have its limitations. The TCP/IP community will soon agree on a selective acknowledgement scheme so that TCP can employ large window sizes efficiently [17]. Selective acknowledgements will reduce Flowsim's packet train length somewhat, because sources will send packets out of order and some acknowledgements may not be compactable at all. Priority scheduling mechanisms may also decrease train lengths and adversely affect Flowsim's performance.

6.3 Future Directions

This paper contributes three techniques that enable us to simulate bigger packet networks. Two of these techniques introduce no approximation and can double simulator performance. We need simulation tools that let us study bigger and bigger networks because network phenomena are difficult to predict. Simulation of a dozen nodes can mislead one to choose a particular adaptive routing algorithm or flow control algorithm. For example, Floyd showed that IP routers tend to self-synchronize and send routing update messages periodically *only* when the number of routers exceeds a certain threshold [12].

We are now employing randomly generated network topologies driven with models of dynamic data and real-time traffic [6] to evaluate various flow control and adaptive routing algorithms.

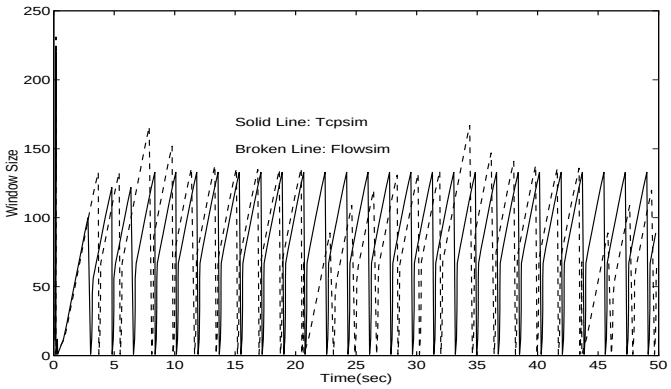


Figure 15: TCP Congestion Window

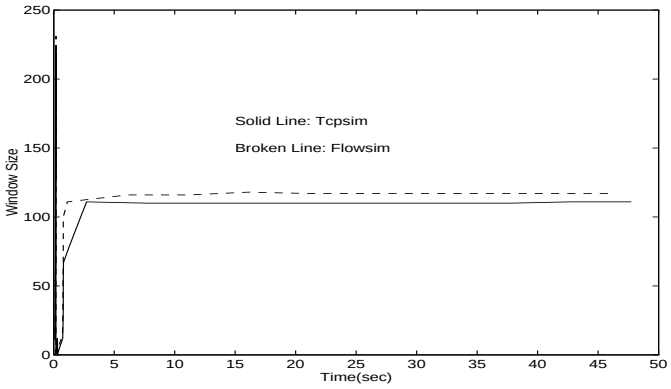


Figure 16: Tri-S Congestion Window

References

[1] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice-Hall, 1987.

[2] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. *ACM SIGCOMM '94*, May 1994.

[3] Randy Brown. Calendar queues: A fast o(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, 1988.

[4] John B. Carter and Willy Zwaenepoel. Optimistic implementation of bulk data tranfer protocols. *1989 ACM SIGMETRICS Conference*, pages 61–69, May 23-26, 1989.

[5] Douglas Comer. *Internetworking with TCP/IP*, volume 1 (2nd edition). Prentice Hall, 1991.

[6] Peter B. Danzig, Sugih Jamin, Ramon Caceres, Danny J. Mitzel, and Deborah Estrin. An artificial workload model of TCP/IP internetworks. *Journal of Internetworking: Practice and Experience*, 3(1):1–26, March 1992.

[7] Lixia Zhang David Clark, Scott Shenker. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *ACM SIGCOMM 92*, August 1992.

[8] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM 89*, 19(4):2–12, August 19-22, 1989.

[9] Julio Escobar and Craig Partridge. A proposed segmentation and re-assembly (SAR) protocol for use with asynchronous transfer mode (ATM). *IFIP WG6.1/WG6.4*, November 27-29 1990.

[10] Deborah Estrin and Danny Mitzel. An assessment of state and lookup overhead in routers. *IEEE Infocom '92*, pages 2332–2342, May 1992.

[11] A Dupuy et al. NEST: A network simulation and prototyping testbed. *Communications of the ACM*, 33:10:63–74, Oct 90.

[12] Sally Floyd and Van Jacobson. The synchronization of periodic routing messages. *SIGCOMM 93*, pages 33–44, October, 1993.

[13] Sally Floyd and Van Jacobson. Traffic phase effects in packet-switched gateways. *Journal of Internetworking:Practice and Experience*, 3(3):115–156, September, 1992.

[14] Steven A. Heimlich. Traffic characterization of the NSFNET national backbone. *Proceedings Winter USENIX Conference*, pages 207–227, January 1990.

[15] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM 88*, pages 273–288, 1988.

[16] Van Jacobson. Compressing TCP/IP headers for low-speed serial links. Technical Report RFC1144, LBL, February 1990.

[17] Van Jacobson, R. Braden, and D Borman. TCP extensions for high performance. Technical Report RFC1323, LBL, ISI, Cray Research, May 1992.

[18] Raj Jain and Shawn A. Routhier. Packet trains–measurement and a new model for computer network traffic. *IEEE JSAC*, pages 986–995, September, 1986.

[19] Hemant Kanakia, S. Keshav, and Partho P. Mishra. A benchmark suite for comparing congestion control schemes. Technical Report (unpublished), ATT Bell Laboratories, July 1992.

[20] Srinivasan Keshav. A control-theoretic approach to flow control. *ACM SIGCOMM '91*, pages 3–15, September 1991.

[21] Paul E. McKenney and Ken F. Dove. Efficient demultiplexing of incoming TCP packets. *ACM SIGCOMM 92 Conference*, pages 269–279, August 1992.

[22] George Varghese and Tony Lauck. Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility. *Symposium on Operating System Principles*, pages 25–38, 1987.

[23] David B. Wagner, Edward D. Lazowska, and Brian N. Bershad. Techniques for efficient shared-memory parallel simulation. *SCS Multiconference: Advances in Parallel and Distributed Simulation*, pages 29–37, 1990.

[24] Zheng Wang and Jon Crowcroft. A new congestion control scheme: Slow start and search(tri-s). *Computer Communication Review*, pages 21(1):32–43, January 1991.