MIT/LCS/TR-470

RATE-BASED CONGESTION CONTROL IN NETWORKS WITH SMART LINKS

Andrew Tyrrell Heybey

January 1990

Rate-Based Congestion Control in Networks with Smart Links

by

Andrew Tyrrell Heybey

Revised version of a thesis submitted to the
Department of Electrical Engineering and Computer Science
in May 1988 in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering

January 1990

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

This research was supported by the Defense Advanced Research Projects Agency, monitored by the Office of Naval Research under Contract No. N00014-85-K-0125 and monitored by the National Aeronautics and Space Administration under Contract No. NAG2-582.

Rate-Based Congestion Control in Networks with Smart Links

by

Andrew Tyrrell Heybey

Revised version of a thesis submitted to the
Department of Electrical Engineering and Computer Science
on May 16, 1988
in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Abstract

I use a network simulator to explore rate-based congestion control in networks with ''smart'' links that can feed back information to tell senders to adjust their transmission rates. This method differs in a very important way from congestion control in which a congested network component just drops packets--the most commonly used method. It is clearly advantageous for the links in the network to communicate with the end users about the network capacity, rather than the users unilaterally picking a transmission rate. The components in the middle of the network, not the end users, have information about the capacity and traffic in the network.

I experiment with three different algorithms for calculating the control rate to feed back to the users. All of the algorithms exhibit problems in the form of large queues when simulated with a configuration modeling the dynamics of a packet-voice system. However, the problems are not with the algorithms themselves, but with the fact that feedback takes time. If the network steady-state utilization is low enough that it can absorb transients in the traffic through it, then the large queues disappear. If the users are modified to start sending slowly, to allow the network to adapt to a new flow without causing congestion, a greater portion of the network's bandwidth can be used.

Keywords: networks, rate-based congestion control, network oscillation

Thesis Supervisor: David D. Clark
Title: Senior Research Scientist, MIT Laboratory for Computer Science

Acknowledgments

Many thanks to David Clark for all his advice and help. I have learned a great deal while preparing this thesis.

Thanks also to Lixia Zhang and Mark Lambert for answering my questions.

Last but not least, thanks to my mother and father for all of their help, support and encouragement throughout my life, and to my wife Karen for putting up with me.

Table of Contents

Table of Figures

Table of Tables

Chapter One

Introduction

A network consists of users, who want to transmit data to one another, connected by various switches and links. Each component of the network has some finite capacity for processing and/or transmitting data packets. Therefore, the transmission of data in the network must be controlled in two logically separate but necessarily interrelated ways. The sender must not send faster than its corresponding receiver can process the data (flow control), and the sum of the flows through any part of the network must not exceed the capacity of that part (congestion control). Each method of control is limited by the other. The transmitter cannot send faster than the capacity of the network even if the receiver could process the flow, and the network cannot force the transmitter to use all of the available capacity if doing so would overwhelm the receiver.

In this thesis I use simulation to explore a new method of congestion control (from [Mosely 84]) in which the links in the network regulate the rate of the data being sent by the individual users. This introduction gives an overview and describes the limitations of current flow and congestion control methods and describes the operation of the new rate-based method. Chapter 2 contains a detailed description of the network model used in the simulation and of the implementation of the rate-based congestion control algorithm. In Chapter 3, I describe the results obtained by running the simulator on two different network topologies, one designed by Mosely and a simpler one designed by me. I also attempt to obtain better results by modifying the operation of the protocol. Finally, in Chapter 4 I draw some conclusions from the results and give suggestions for further research.

1.1 Flow Control

The sender and receiver of data must somehow agree on the rate and amount of data flowing from the sender to the receiver. Such *flow control* is most commonly accomplished by using windows. The window is the amount of unacknowledged data that the sender is allowed to send. When a receiver acknowledges the receipt of some of the data, the sender can send that much more. At any given time there is only as much data in transit as the size of the window.

Window-based flow control works acceptably well for relatively low delay networks. However, it fails when it is necessary to have large amounts of data in transit in order to utilize all of the bandwidth of the network, as in a high-speed but high delay network. If a data transfer is to make effective use of the available bandwidth, the window must be made very large. The window must be large enough for the sender to keep sending until it gets an acknowledgment from the receiver.

However, if such a large window is transmitted all at once, the flood of packets is very likely to overload some intermediate network component, resulting in the loss of packets.

When using a network with a very large delay, a flow control method based on the regulation of the rate of the data being sent can work much better. Rather than controlling the amount of data outstanding, the sender and receiver negotiate the rate at which packets will be transmitted. The NETBLT protocol [Clark 87] uses rate-based flow control. NETBLT has been implemented and shown to be able to use a large fraction of the available bandwidth even over a long-delay network such as the Wideband satellite network. The Wideband network has a round-trip delay of 1.8 seconds, and an available bandwidth of approximately one megabit/second. In tests, NETBLT achieved steady-state (not including time to set up the connection) throughput values between 926 and 942 megabits/second, using more than 90 percent of the available bandwidth [Lambert 87].

1.2 Congestion Control

While flow control concerns the ends of a single connection, *congestion control* concerns the aggregate of the flows in the network. When the sum of the flows through a component of the network exceeds its capacity, something must give. Usually, when a network becomes congested, the result is a build up of packets in the queues of its switches. The switches either cannot process the packets fast enough, or the links connected to the switches cannot transmit the packets fast enough. In either case, switches most commonly deal with congestion by dropping packets when they reach the limit of their buffer capacity. In a system using window-based flow control, the build up of queues and dropping of packets causes the users to reduce the amount of data sent because acknowledgments will be delayed or lost because of the congestion.

In contrast, the loss of packets does not slow down NETBLT, which uses rate-based flow control. The sender will keep sending packets at the negotiated rate (at least until a complete buffer of data is sent; see the NETBLT paper) regardless of how many are lost in the network. The end users negotiate the transmission rate without any knowledge of the intervening network. (Of course, some knowledge about a workable rate must be either hardwired into the protocol or given by the (human) user, since the protocol as described in [Clark 87] has no way of finding out from the network what sort of a rate is reasonable.) Some work is being done on an algorithm for NETBLT that uses the interpacket arrival time to decide, without any direct communication from the network itself, if there is congestion or not. Since the receiver knows the rate at which the packets were transmitted, if they arrive at a different rate there must be some sort of congestion along the way. If congestion is detected, the transmission rate is adjusted to the rate at which the packets were actually received [Lambert 88]. While this method of congestion detection and control has worked in simulation, it does have one disadvantage. It is possible to end up with an unfair allocation of bandwidth among the various flows through the congested network component. If the switches that are congested in the network control their flows, they can allocate a fair proportion of the available bandwidth to each flow.

This thesis explores the behavior of a simple computer network using another type of rate-based protocol. The end users send data at some rate which is determined by feedback from the links in the network. The links in the network calculate a ''control rate'' based on the number of flows using the link, capacity of the link, and current transmission rate of the various flows through the link. The control rate of a link is the maximum rate, in bits per second, at which any sender using this link should transmit. As a packet travels through the network, each link puts the minimum of the packet's current control rate and the link's calculated control rate into the packet (the sender sets the control rate of the packet to infinity when it is generated). Therefore, when the packet reaches its destination, it contains the minimum control rate of all the links along its path. The receiver periodically transmits short control packets to its sender that contain the last control rate received. When the sender gets a control packet, it sets its transmission rate to the control rate found therein, thereby completing the feedback loop.

I simulate three algorithms from Mosely for calculating the control values. Mosely examined the convergence and fairness of the three algorithms, and ran a computer simulation of them. With a dynamic network load, her simulation produced unsatisfactory results for all three. The links could not prevent changes in the number of flows from causing large packet queues to build up. She concluded that the control value algorithms did not converge quickly enough to handle the dynamics of the packet voice model that she was simulating.

The protocol exhibits an interesting non-linearity as successively larger loads are placed upon it. As the applied load grows closer to the link capacity, the queues jump from tens to thousands of packets. There is essentially no intermediate region of operation. Once the queues become so large, the delay becomes so huge as to render the feedback control useless.

The protocol can use more of the link capacity before failing by introducing a ''slow start'' convention.[1] A new talk spurt sends at one tenth of its assigned rate until it receives its first control packet. This delay gives the links a better chance to react to the new flow before its data can seriously affect the network. Slow start does not prevent the instability--it merely moves the point of instability higher on the scale of link utilization.

[1]This ''slow start'' shares its name and general idea with Van Jacobson's slow start for TCP ([Jacobson 88]). They are otherwise unrelated.

Chapter Two

The Simulation

2.1 The Model

I used Mosely's network model adapted to my simulator to perform the experiments. The network consists of users, hosts, switches, and links. Hosts and switches did not exist in Mosely's simulator, and merely contain the packet queues in my simulator (see below). Users transmit data packets to their partners (other users) at a constant rate. The links along the route of a packet use information about the sender's transmission rate contained in the packet to keep statistics about the number of flows in the link and the transmission rates of those flows. Using the table of flows and rates, each link calculates a control rate at which the flows through it should send. This rate is inserted into packets that pass through the link, and the receiver of the packet periodically puts the last received control rate into a control packet and sends it back to the sender. The sending user changes the length of the data packets that it sends to make its data transmission rate be the rate as received in the latest control packet. This feedback loop works in both directions. In other words, the data packets arriving at the receiver contain a transmission rate for the receiver (to use when it starts sending) just as the control packets arriving at the sender contain the rate for the sender. Figure 2-1 shows the structure of a path that data packets might take to get from sender to receiver.

The statistics of the flow of packets in the network are also from Mosely, and model the packet-switched transmission of voice. Each user sends data only to one other user, its partner. The partners alternate sending data packets to each other in "talk spurts." The length of each talk spurt in packets is a geometric random variable. Since the *packet* transmission rate is constant, the *data* transmission rate is varied by varying the length of the packets. When a talk spurt ends, the sending user marks the last packet as such and becomes the receiver. When the receiver receives the last packet it becomes the sender. A detailed description of each network component follows:

2.1.1 USER

Users come in pairs. Each user sends to and receives from its partner only. The partners alternate sending data packets in talk spurts that last (on the average) 1.2 seconds. An active user sends 50 data packets per second. The data transmission rate is varied by making the length of each packet be the desired rate (in bits/second) divided by 50. Each packet sent by a user contains a control rate, a feedback rate, and the current transmission rate of the sending user. The control rate is initialized to infinity when the packet is created, and is modified by links along its route as

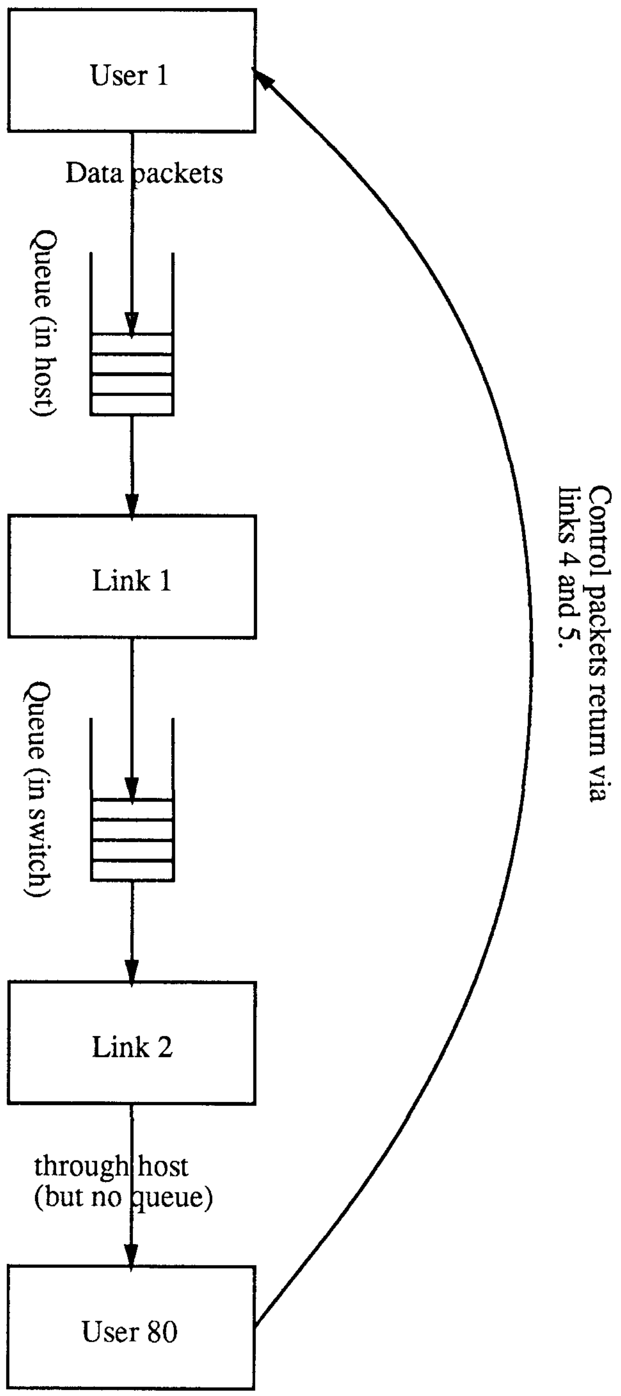

Figure 2-1: Example path between two users

discussed below. The feedback rate is the control rate from the last control packet received (which will become the current transmission rate of this user's *partner*). A flag in the packet is set if it is the last packet in a talk spurt. Each data packet has a 1/60 probability of being the last packet in a spurt, so that the number of packets per talk spurt is a geometric random variable with mean 60.

The inactive partner in a pair of users sends a control packet to its active partner every 100 milliseconds. The control packet has a fixed length of ten bits, and contains only the control rate and feedback rate as discussed above.[2]

When a user (active or inactive) receives a packet, it sets its current transmission rate to the feedback rate contained in the packet (which is the control value last received by this user's partner), and sets its control value to the control value in the packet (to be sent back to its partner). If the end-of-spurt flag is set, the receiving user becomes active and starts sending data packets.

2.1.2 HOST

A host serves as the intermediary between users and links. It decides where to send packets by looking up their destination in a routing table, then sends the packet on after a constant delay. If the packet is going to a link and the link is busy, the host queues the packet until its destination link is free. There is a separate output queue for each link attached to the host. The host has an infinite number of buffers, and so no packets are ever dropped due to lack of buffer space.

2.1.3 SWITCH

A switch is very similar to a host, but connects only links together, not users.[3]

2.1.4 LINK

A link receives a packet from a host or a switch, and then transmits the packet to the next host or switch on the packet's route. (The next host or switch on the route has been placed into the packet after the routing table lookup in the previous host or switch.) The time to transmit the packet is a constant plus the link's capacity divided by the packet length. The minimum of the control value in the packet and the link's current control value (calculated as described below) is placed in the packet.

[2]The ten bit length is an arbitrary limit used by Mosely. I am not sure how to fit both of these values (not to mention destination address, etc.) into ten bits in reality. Also, the 100 millisecond interval between control packets is the number chosen by Mosely as often enough, but not so often as to add a significant load to the network.

[3]Mosely did not have hosts or switches. A link's queue was contained within the link, and each packet contained its route. My implementation behaves similarly to hers; this representation conforms to the structure of the simulator used for this evaluation.

Each link maintains a table, keyed by destination user, containing the current transmission rate of each flow as reported in the last data packet received belonging to that flow. When the last packet in a talk spurt is received, the flow is removed from the table. When a packet for a destination not in the table is received, a new entry is created in the table. At fixed intervals, the link uses the information from this table to calculate a new control value, using one of the three algorithms described below.

2.1.5 Routing

Routing in the network is static. A route is defined in each direction for each pair of users (so the path from one user to its partner may be different than the return path). The routes are stored in a global table, and each host or switch uses itself and the packet's destination as a key in order to find out the next link and next host or switch in the packet's route.

2.2 Control Value Algorithms

The links calculate their control values using one of three formulas, (using the following variables):

$p_j(t)$	Control rate (in bits/second) of link number j at time t. This value is placed into each packet passing through the link (as described above) so that it will find its way back to the transmitter of data packets.
c_j	Effective capacity of link j, $c_j = aC_j$, where $0.0 < a < 1.0$ and C_j is the real capacity of the link in bits per second.[4] Used to calculate the control value as described below.
W_j	Current number of flows seen at link j. This number is the size of the link's table of flows and transmission rates as described above. When a packet is received that belongs to a flow that is not in table, W_j is incremented. When the last packet in a talk spurt is processed by the link, its transmission rate is removed from the table, and W_j is decremented.
r_{ij}	Current transmission rate of flow i as seen at link j. This is the current transmission rate as reported by each active user in its packets.
f_j	$\sum_i r_{ij}$ for each active flow i through the link. This is the reported total flow of data through the link, calculated by adding the reported rates of all flows through the link.

The three formulas:

Hayden: $p_j(t+1) = \max[c_j/W_j, \min[c_j, p_j(t) + (c_j - f_j(t))/W_j]$.

The new control rate is the old control rate adjusted up or down by the amount of extra or deficit capacity. The adjustment is divided by the number of flows since the control rate applies to each flow. This formula is from [Hayden 81].

[4]Mosely fixed a at 0.8, while I used several different values. See Chapter 3.

Mosely: $p_j(t+1) = \max[c_j/W_j, \min[c_j, \max[r_{ij}(t)] + (c_j - f_j(t))/W_j]]$.

$Max[r_{ij}(t)]$ is the maximum transmission rate reported of all the flows though the link. The new control rate is the largest reported transmission rate adjusted up or down by the amount of extra or deficit capacity. The adjustment is divided by the number of flows since the control rate applies to each flow. The largest reported transmission rate is used so that if some of the flows are constrained by other, slower (or more busy) links then the flows that *are* constrained by this link will get more of this link's capacity.

Jaffe: $p_j(t+1) = \max[c_j/(W_j+1), \min[c_j, p_j(t) + (c_j - f_j(t) - p_j(t))/(W_j+1)]]$.

The new control rate is the old control rate adjusted up or down by the amount of extra or deficit capacity minus the old control rate. The control values calculated by this formula tend to be smaller than those calculated by the other two for two reasons. The effective capacity of the link used in the calculation is smaller. First, it is divided by $W_j + 1$ rather than W_j. Second, the current control rate is subtracted from the adjustment. This formula is from [Jaffe 80].

Note that each of the formulas bounds the control value by the effective capacity of the link divided by the number of flows (number of flows plus one for Jaffe) and the effective capacity.

The links calculate a new control value either every 20 or every 100 milliseconds.[5] In addition, the links can optionally use a "protocol" (invented by Mosely) to decide whether to calculate a new control value or not. The idea behind the protocol is to wait until all senders have received the latest control rate before computing a new control rate. The protocol allows a new value to be calculated only if all the flows currently in the link's table are transmitting at a rate less than or equal to the link's current control value: $r_{ij}(t) \leq p_j(t)$ for every flow i through the link. This requirement guarantees that all senders using the link have either received the new rate, or are transmitting at a lower rate received from some other link. If the protocol is not satisfied, the link waits for 20 milliseconds before trying again.

2.3 The Simulator

The simulator that I used is one developed at MIT. It is an interactive, event driven simulator. It consists of an I/O manager that provides an interface to the X window system, and an event manager that allows network components to schedule events for each other. Any sort of packet switching network component can be simulated, provided that its behavior can be described in a single-threaded C program. I wrote new components to act as users and links, but used existing host and switch components to connect users and links together, to route the packets, and to manage the packet queues for the links. All simulations were run with identical random number generator seeds.

[5]These parameters can all be set to any value--the numbers given here are just the ones that Mosely and I used in the simulation.

Chapter Three

Results

3.1 Mosely's Topology

Mosely's topology consists of eight links and 80 users. (See Figure 3-1.) Each user *i* has as its partner user 81–*i*. Links one through four handle 20 users each (averaging 10 flows each, since only half the users are active at any given time). Links five through eight handle eight, 16, 24, and 32 users (an average of four, eight, twelve and sixteen flows) each, respectively. The links are unidirectional, and in general the packets of each partner travel along different links.[6] For example, user 1 transmits packets to its partner, user 80, over link 1 followed by link 8. User 80 sends packets back on link 4 followed by link 5.

Mosely ran simulations for both a static and dynamic network. In the static network, all the even numbered users transmit without stopping. The length of a talk spurt is infinite. She showed that all of the different algorithms eventually converge to the correct control value, though with different types of oscillation and different periods. In the dynamic network, the users alternate talk spurts as described above. Mosely simulated each of the algorithms with and without the protocol. The Hayden and Jaffe algorithms were run only with slow control rate updates, and the Mosely algorithm only with fast updates because she concluded from the static results that they performed best under those conditions.

In Mosely's simulation, none of the algorithms behave very well, and in particular the Hayden and Mosely algorithms ''produce totally unacceptable queues.''[7] As shown by the static results, every algorithm takes at least 2-3 seconds to converge. Mosely concludes that the algorithms do not work well in the dynamic case because all of the algorithms take too long to converge to the correct control value.

The rate at which the control rates converge is not the problem *per se*. The problem is that the new flows enter while transmitting at a control rate calculated for a smaller number of flows. Until the new control rates are fed back to the senders, the link will be overloaded. If the link does not have enough unused capacity to absorb the transient, large queues will build up regardless of the control rate calculation algorithm. Mosely does not run the dynamic simulation for Hayden or Jaffe with fast control rate updates, nor with her own algorithm with slow updates

[6]Four pairs do share link 7, but they should not make a significant change in the operating characteristics of the network.

[7]Mosely, p. 131.

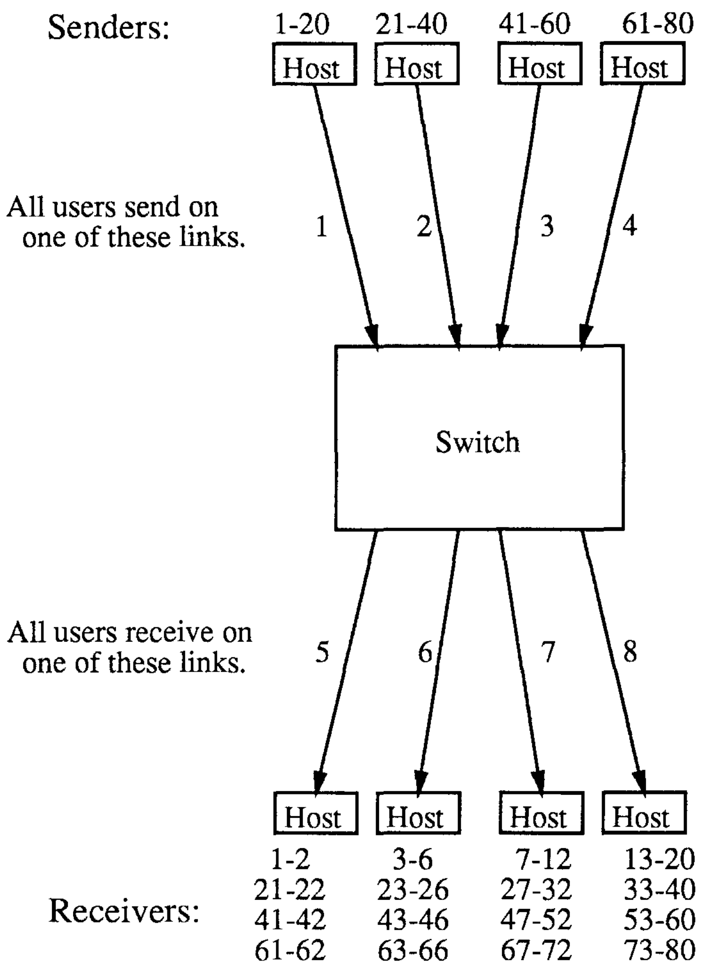

Figure 3-1: Mosely's Topology

(or at least she nowhere mentions such a simulation). She bases this decision on the static results. I ran all permutations of algorithm, update frequency, and protocol. The variations that Mosely did not run are neither better nor worse than the others--they do not follow the static results. Just as an example, Hayden's algorithm with no protocol and slow update control updates, (figure A-1) (one that Mosely did run) certainly does not behave any better than the same algorithm with fast control rate updates (figure A-2).

She also comments that the ''bits transmitted'' value for the Jaffe algorithm oscillates, and that the oscillation is because of the oscillation in control value in the static case. However, the value does not oscillate on its own--it actually follows the number of flows. As the number of flows decreases, the bits transmitted rate will go down until the control value changes to compensate, *and* the change has been fed back to the active users. In some sense, the bits transmitted value is the derivative of the number of flows. The only reason that the Jaffe appears to ''oscillate'' more than the other algorithms is that it does a better job of avoiding large queues. If there is a queue of packets waiting to be transmitted over a link, the bits transmitted value will naturally stay at the capacity of the link, and not oscillate at all. If the packet queue is generally empty, however, the bits transmitted value will vary as the number of flows changes, as described above.

Finally, she concludes that Jaffe is the best of a bad lot. The queues that build up are not as big as the others, and the average delay between the time a packet enters the queue and the time that it finishes transmission is much smaller than any of the other algorithms. In addition, the throughput of Jaffe is only slightly lower than the others. However, delay is not a valid comparison between these algorithms in this case. Of course the other algorithms have a much greater delay--the packets spend much of their time waiting in queues. In this case, delay is a symptom, not a problem with the algorithm. Second, the only reason that Jaffe does not have such large queues is that it does not attempt to use as much of the capacity of the link as the other two. When calculating a new control value, it uses W_j+1 as the divisor rather than W_j. The effective capacity of the link is therefore smaller than the other two. As I will show later, if the effective capacity is lowered for the other two algorithms, they will not have large queues either, while if the effective capacity is increased for Jaffe, it does not work any more. Finally, *if* large queues build during a simulation run, the average throughput is also not a valid comparison. During the time that there are packets in the queue, the link should be running at its maximum capacity. However, this throughput, while nice and high, is not in any sense a measure of the effectiveness of the particular algorithm. The circumstances under which the large throughput is obtained (large queues) are undesirable.

3.2 My Topology

In order to more easily look for new explanations for the poor behavior of the algorithms, to look at the interaction of traffic in two directions, and to explore the oscillation problem, I ran more simulations with a simpler topology. I also ran the simulations for 150 seconds rather than 30 to see how the network behaved for longer periods of time, and varied the effective capacity parameter.

The topology I used has forty users and two half duplex links (the equivalent of one full duplex link). See figure 3-2. All the odd numbered users (1, 3,..., 39) communicate with their partners through link 1, and their partners (user *i* has as its partner user 41–*i*) send packets back through link 2. During normal operation, there should be 20 flows active at any instant (on the average, 10 through each link).

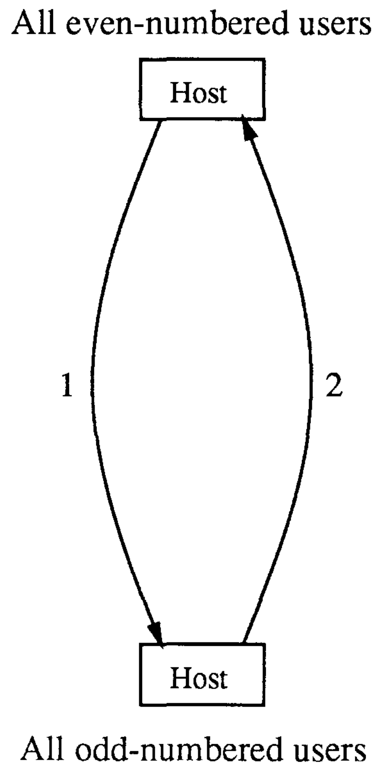

Figure 3-2: Heybey's Simple Topology

3.2.1 Stable and unstable region

The protocol operates either stably or unstably. As the ''effective capacity'' parameter is varied, the maximum queue length observed during a simulation run makes a sudden jump from less than

100 to several thousand. When operating in the stable region, the queues have a startup spike of up to several hundred, then drop to around five packets with occasional spikes up to 50. In the unstable region, the queues oscillate wildly from zero to two or three thousand packets, and down again. The difference can be clearly seen in figures A-5 and A-6. The parameters of the two simulations differ only in the effective capacity, which is 0.55 for the first run and 0.56 for the second. In figure A-5, the queue hovers around ten or so and the number of flows oscillates around 10, while in figure A-6, the queue varies from zero to 2000 and the number of flows oscillates from zero to 15 in phase with the queue oscillation. One run has a region of operation during which the queues oscillate between zero and seven or eight hundred packets. However, the oscillations disappear and the protocol remains in the stable region for the rest of the run.

As the effective capacity parameter is increased, there is not always a clean break between stable and unstable operation. If effective capacity is much lower or much higher than the point at which unstable operation begins, the protocols always operate stably or unstably, respectively. As the effective capacity is increased in steps, the protocol may operate stably, unstably, and then stably again. However, there is no intermediate region of operation. No combination of parameters results in a queue that could be considered to be ''between'' the stable and unstable regions.

Once the queues build up to unreasonable lengths, the packets from which the links get their rate information are so old that the control has no effect. Because of the wild oscillations, the users who have just finished a talk spurt are sending control packets to their partners with rates that are very fast. The link with no queue has no load, and so has set its control rates all the way up.

3.2.2 How to Operate Stably

The key to avoiding the unstable region is to avoid attempting to use too large a percentage of a link's capacity. When running the simulator with the same parameters as used by Mosely, the only algorithm that works at all is Jaffe. As mentioned above, I believe that the reason that Jaffe seems to perform better is because it does not attempt to use as much of the link's capacity as Hayden or Mosely. In order to test this hypothesis, I ran the Hayden and Mosely algorithms with a lower effective link capacity.

Note that the ''effective link capacity'' as set in the simulation and used to calculate the control rates can not be compared between different algorithms. When I adjusted the effective capacity so as to make all three algorithm work, the average throughput (calculated over the entire simulation run) is hardly different between the three algorithms despite the different values for effective capacity. Maximum throughputs are 30 kbps for Hayden (at an effective capacity of 0.7), 31 kbps for Jaffe (at 0.79), and 31 kbps for Mosely (at 0.69).

3.2.3 Slow Start

Since it is the variance in the talk spurts that cause the protocol to enter the unstable region, I tried making the users start a talk spurt by sending slowly. When a user starts a talk spurt, it sends at a rate that is one tenth of its control rate until it receives its first control packet. After receiving its first control packet, it resumes transmitting at full speed. While in slow start, it still puts the full rate in each transmitted packet, but it doesn't actually transmit at that rate.

Slow start increases by a significant amount the percentage of the link capacity that can be used before the protocol enters the unstable region. With slow start, the maximum throughputs (at effective capacity) achieved are 35 kbps for Hayden (0.88), 36 kbps for Jaffe (1.01), and 35 kbps for Mosely (0.82). See Table 3-1 for a complete listing. The table lists the highest effective capacity (and throughput) at which each combination of algorithm and parameters could be run before becoming unstable. In other words, the simulation run with the next larger effective capacity value was unstable. However, once it has become unstable, slow start does not help the protocol recover. The huge queueing delays render any sort of control, slow start or not, worthless.

			with slow start	
Algorithm/protocol/update rate	Eff. capacity	Throughput	Eff. capacity	Throughput
Hayden/no/fast	0.55	25.4	0.70	29.4
Hayden/no/slow	0.70	30.4	0.90	35.7
Hayden/yes/fast	0.39	19.4	0.53	23.8
Hayden/yes/slow	0.58	25.6	0.79	31.7
Jaffe/no/fast	0.73	29.8	0.90	33.4
Jaffe/no/slow	0.81	32.0	1.01	36.2
Jaffe/yes/fast	0.51	22.1	0.70	27.6
Jaffe/yes/slow	0.61	24.3	0.80	29.0
Mosely/no/fast	0.69	32.1	0.87	37.0
Mosely/no/slow	0.73	33.4	0.79	33.3
Mosely/yes/fast	0.70	31.2	0.88	35.8
Mosely/yes/slow	0.74	32.5	0.91	36.4

Table 3-1: Table of maximum throughputs (KBits/sec) and effective capacities

3.2.4 Why the protocol does not recover

When the system does enter the unstable region, it does not, in general, ever recover.[8] There are three problems that contribute to its staying there. One, the convoying effect, is easily visible on all of the unstable figures, but especially on figures A-3 and A-4. Note that the size of the queue and the number of flows through the link oscillate synchronously, and that the number of flows

[8]There are several runs in which the queues oscillated for a few cycles up to 800 or 900 packets and then do recover.

recorded by the link drops close to zero when the queue drains. While there is a large queue on one of the two links, the other link is almost completely unloaded. When the queues become large, the queueing delay becomes longer than the 1.2 second average length of a talk spurt. Therefore, all the active users transmitting through the overloaded link have finished their talk spurts, but their partners have not become active yet since they have not received the end-of-spurt packets which are waiting in the queue. When the queue drains (since there are no more packets arriving in it any more), many of the inactive user start transmitting. Since there were almost no flows in the other direction, they all start transmitting at the very high rate fed back by the unloaded link. A queue then develops on that link, beginning another cycle. Although the flows are supposed to start and stop randomly and independently, they instead fall into a pattern of all the flows going in one direction, then the other.

The second problem that aggravates the instability is the ''forward queueing problem.'' Links extract the sending rate information from the packets as they are transmitted. If there is a large queue in only one direction, however, the information in the packets that the link sees is old. The control value calculated by the link is fed back to the senders very quickly (because the opposite direction has very little load). However, the control is based on old information, and so will not be correct.

I attempted to solve the forward queueing problem by changing the simulator such that the link took the sender's rate out of packets when they were put on the end of the queue, not when they were sent. This way, the link's picture of the world is up to date. Unfortunately, this change resulted in slightly different behavior, but did not solve the problem of large queues. See figures A-7 and A-8. The new strategy led to more and faster oscillation of the queues. The change helps the forward queueing problem--the queues are somewhat smaller, and drain more quickly--but the convoy problem still remains. The senders get a more correct control value, but the inactive users get the large control value from the lightly loaded link more quickly than before. Even more of them start transmitting at the fast rate, and the queue in the other direction builds up much more quickly.

Finally, if there is an extremely large queue in one direction, some of the talk spurts in the other direction will operate without any control at all. The queueing delay of a 2500 packet queue is longer that the duration of a talk spurt. All the control packets from the receiver will be waiting in the queue. Therefore, if the link on which the data is being sent was initially unloaded, the sender will send at its maximum rate for the entire duration of the talk spurt. If only a few senders start under these conditions on the unloaded link, they will build up enough of a queue to continue the oscillation as the other queue drains and more sender begin talk spurts.

Chapter Four

Conclusions

The key to congestion control with these rate-based algorithms is to insure that the network can absorb the transients in the data flow expected in the environment. Any of the three algorithms can provide effective congestion control with almost equal throughputs if the ''effective capacity'' used in their control value calculations is adjusted correctly.

If the effective capacity is not adjusted correctly, and large queues do build up, then none of the three algorithms will in general recover. The convoying and forward queueing problems collaborate to cause the huge packet queues to oscillate from link to link, at least for this particular set of traffic dynamics.

The effect of the convoy problem is that the flows all travel in one direction, then the other, rather than in random patterns as intended. With a large queue, all of the currently active flows in the network end before the end-of-spurt packets drain out of the queue. Since the flows have ended, the link without a queue has little or no load on it. When the end-of-spurt packet do finally arrive at their destinations, all of the inactive users start sending over the previously unloaded link at the maximum rate. That link then suddenly has an attempted flow of four or five times its capacity, a large queue accumulates, and the cycle starts all over again.

The forward queueing problem is also because of the large queue. The link updates its information about a flow and its current transmission rate when it sends a packet of that flow. If there is a large queue, the packet is very old, so the link is working with old information. Therefore, the control values that are being produced and sent back to the senders are based on what the senders were doing a second ago or more.

I tried to alleviate the forward queueing problem by giving the links the information in a packet when it arrived on the queue, not when it was sent. However, this attempt aggravated the convoy problem. The fix, while helping the queue drain more quickly, also ensured that the large control value of the relatively unloaded link was quickly sent to the inactive users so that even more of them transmitted at the large rate when they became active.

I also tried to lower the percentage of the bandwidth that had to be reserved to absorb transients by changing the users to use slow start when beginning a talk spurt. Slow start allowed me to set the effective capacity of all the algorithms higher (resulting in a higher average throughput) before they slipped over the edge into instability.

Appendix A

Figures

All of the figures in this appendix depict the number of bits transmitted through the link every 0.1 seconds (yielding a number approximately equal to the bits per second throughput of the link divided by 10), the current number of flows seen by the link times 100, and the output queue size, in packets, of the host or switch attached to the link. The queue size is scaled either by 10 or by 1, depending on its maximum value (if the captions contains ''Qx10'', the queue length graph has been scaled). Since the bits transmitted value is calculated every tenth of a second, it will not always be exactly accurate. A packet whose transmission crosses a tenth of a second boundary will be counted entirely in the second interval, making the value too low in the first interval and too high in the second one.

Figures A-1 and A-2 (the next two pages) are from my simulation of Mosely's topology and her set of parameters. Link capacity is 40 kbps, effective capacity is 0.8 times real capacity, propagation delay is 3 milliseconds, and link update attempt interval (the time between attempts to calculate a new control rate) is 20 milliseconds. The algorithm, update interval, and protocol use are as shown in the figure captions.

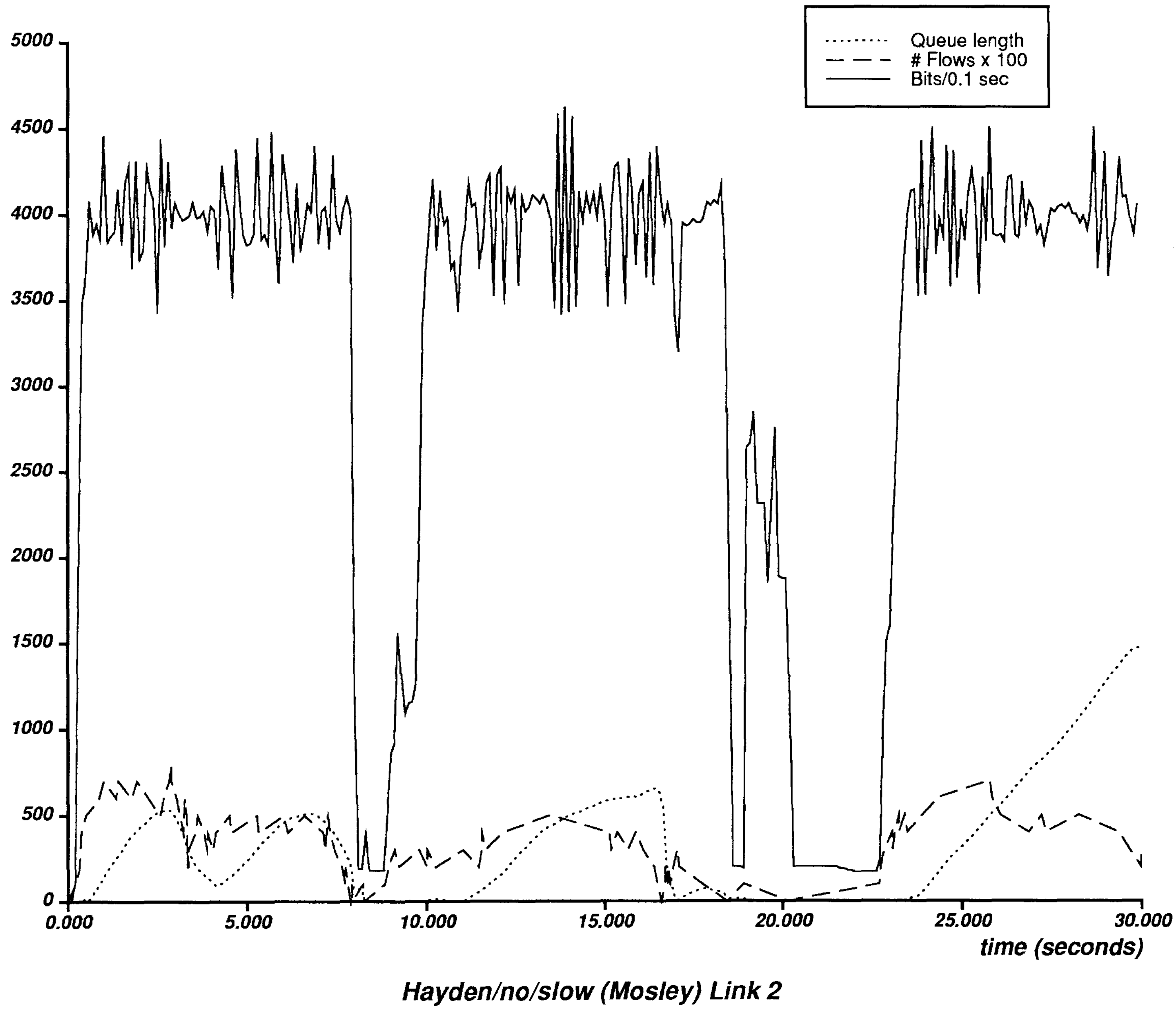

Figure A-1: Mosely's Topology 1

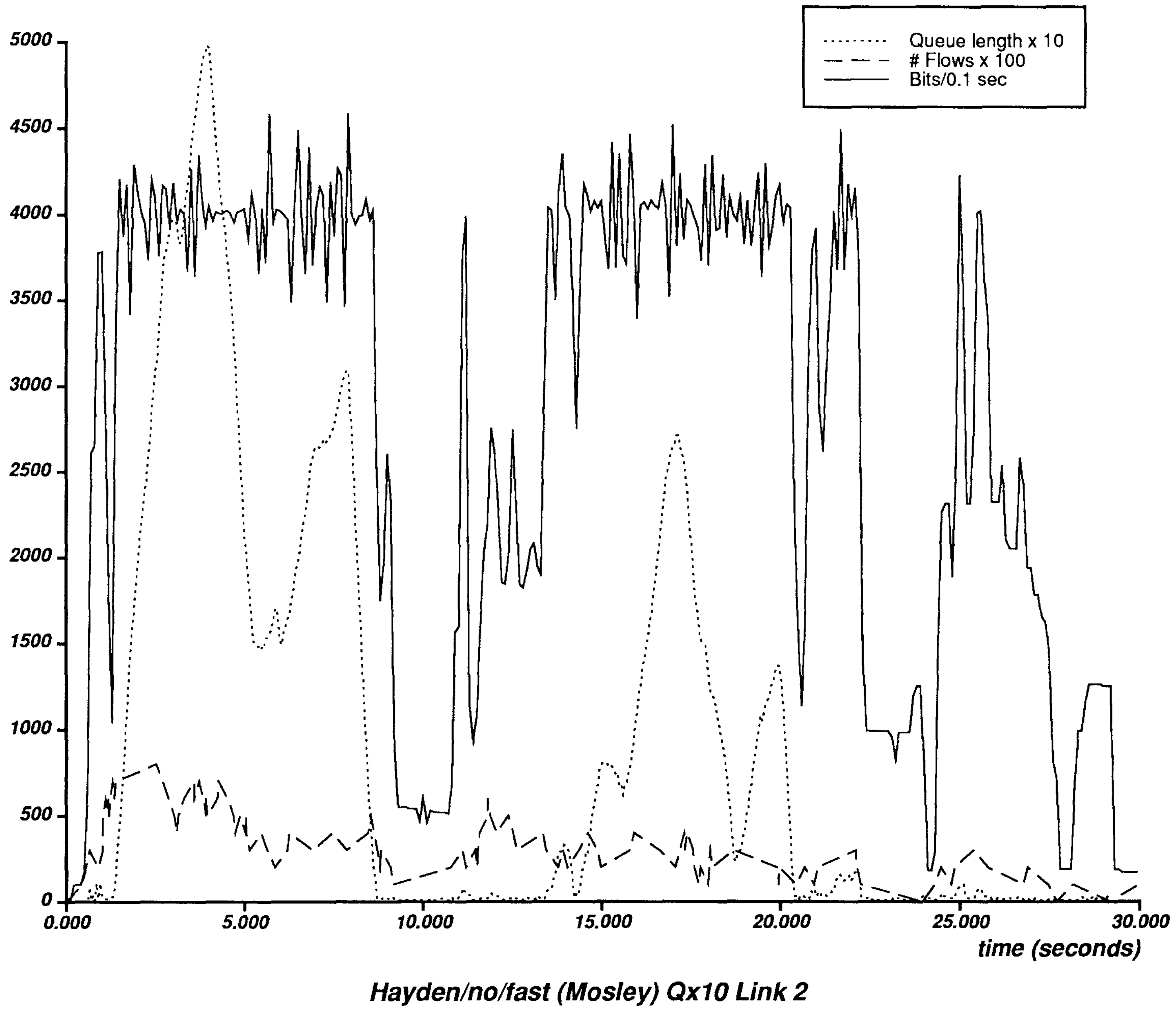

Figure A-2: Mosely's Topology 2

The next two figures (A-3 and A-4) were produced by running my simple topology (see figure 3-2) using all of Mosely's parameters unchanged (as described above). The character of the results (compared to figures A-1 and A-2) is not changed because of the different topology (note that the time scales of the two pairs of figures are different).

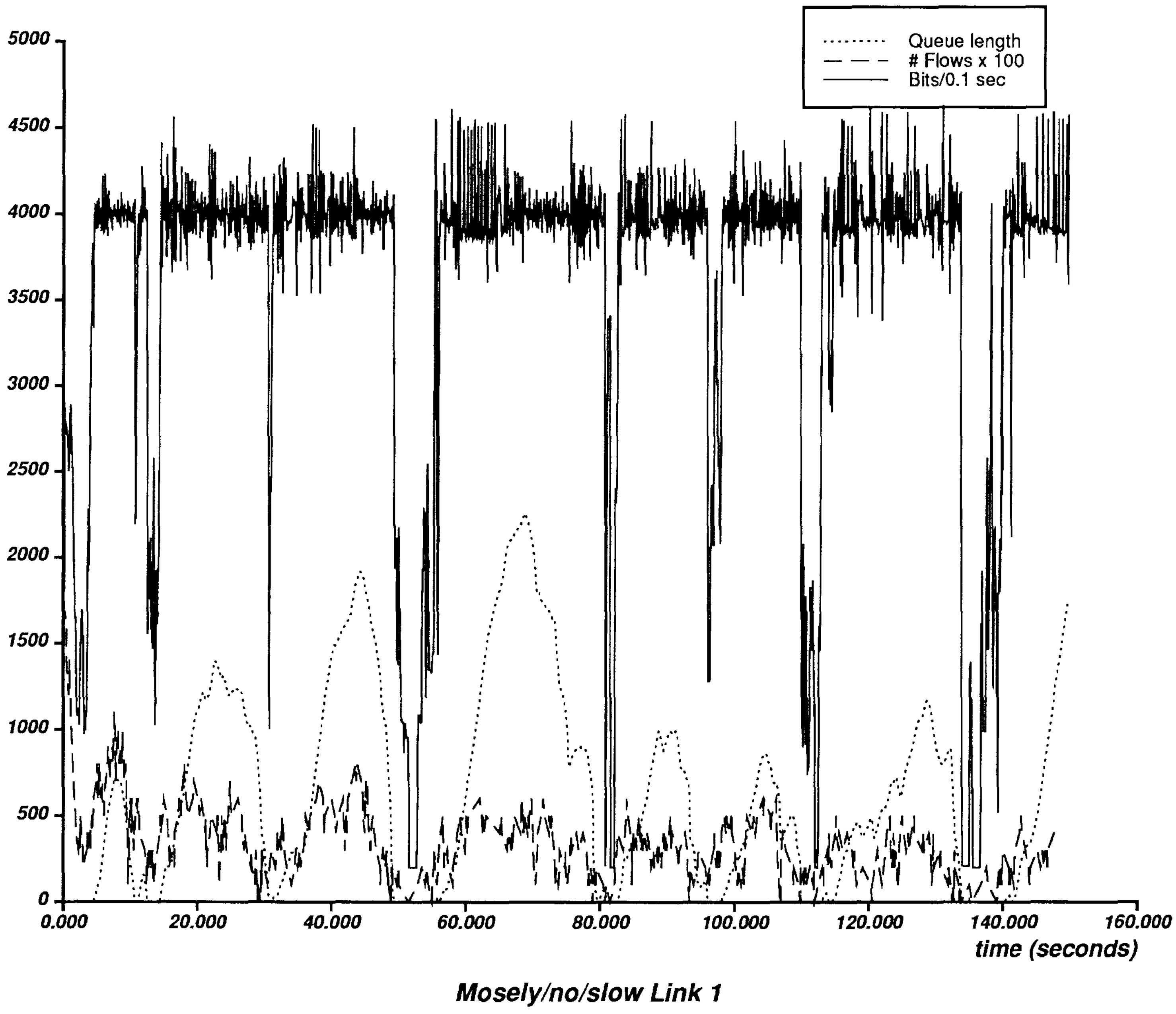

Figure A-3: Heybey's Topology, Moseley's Parameters 1

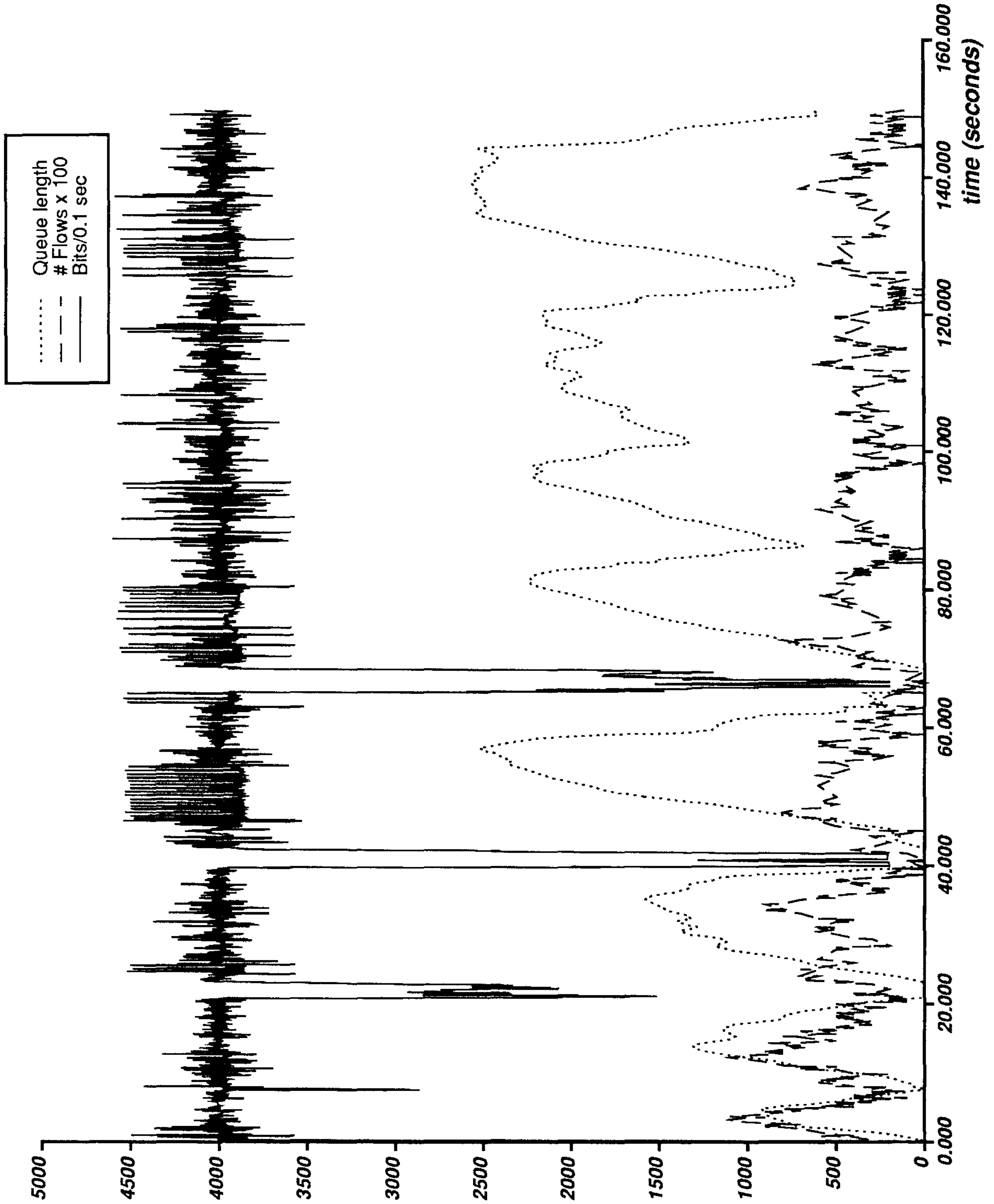

Figure A-4: Heybey's Topology, Moseley's Parameters 2

The next two figures (A-5 and A-6) are examples of the difference between stable and unstable operation of the protocols. The particular examples were run on Heybey's topology using Hayden' algorithm, fast (20 millisecond) updates of the links' control rates, and not using Moseley's "protocol." The parameters of the two simulations differ only in that that figure A-5 used 0.55 as the effective capacity, while figure A-6 used 0.56.

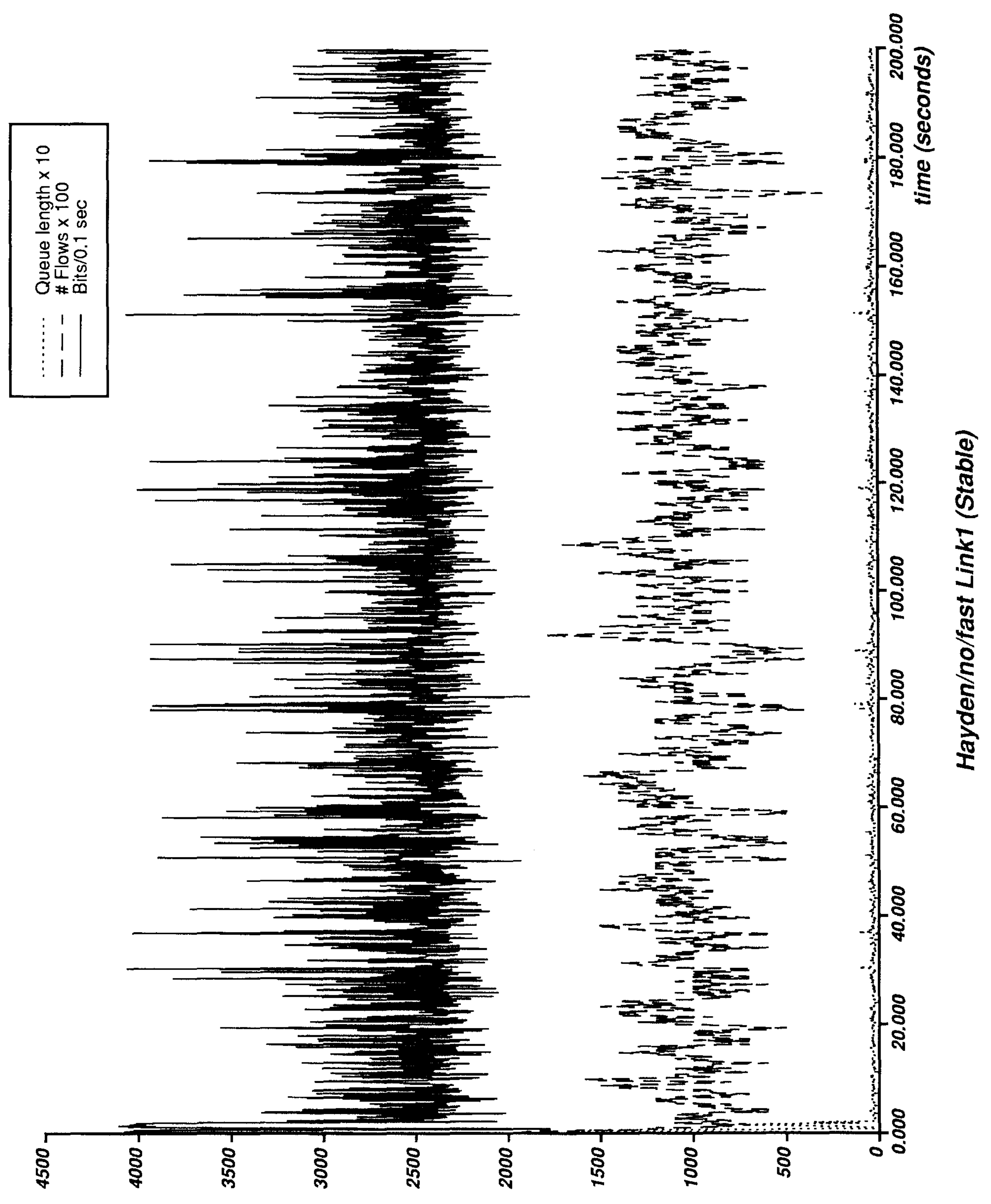

Figure A-5: Stable Operation

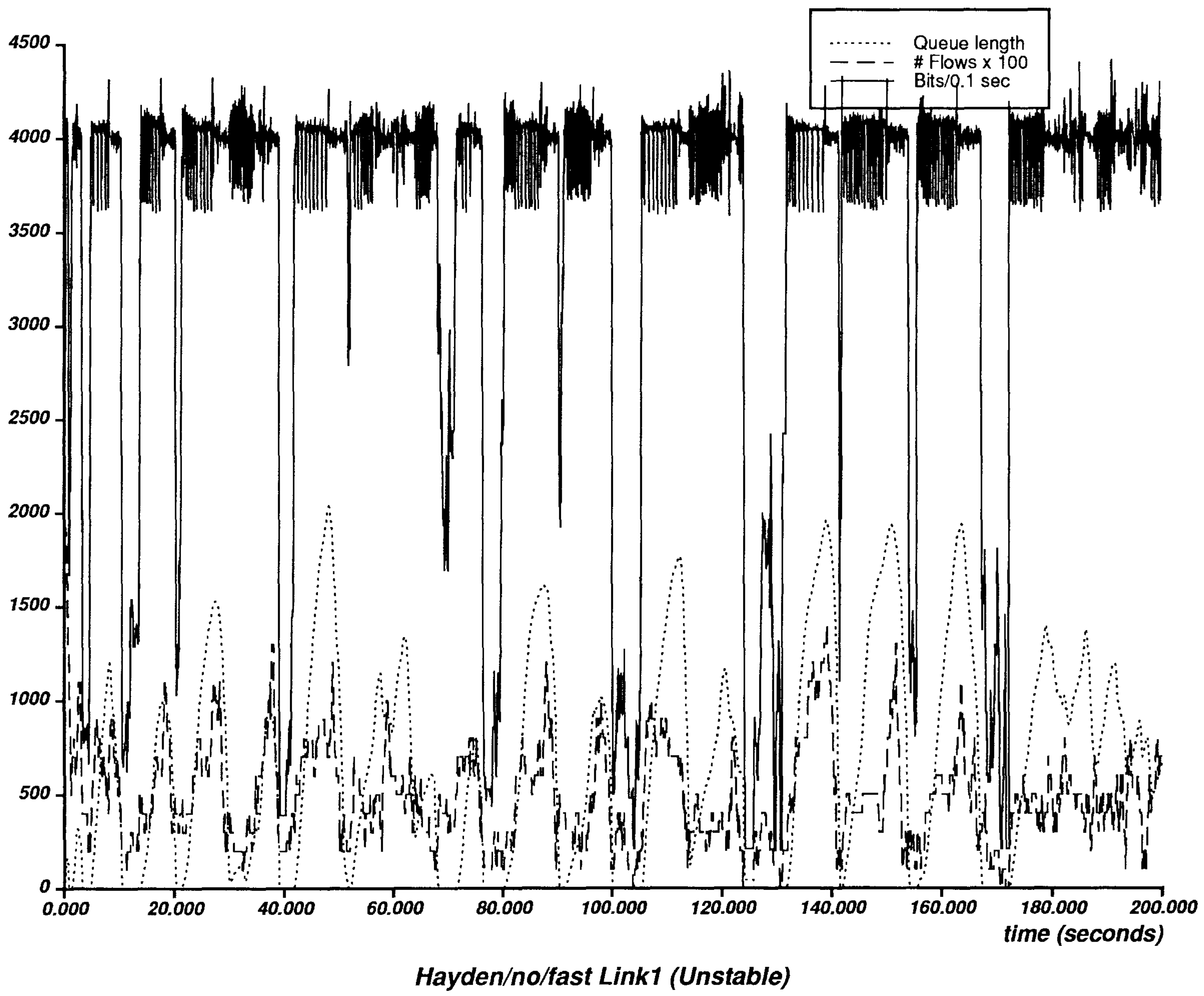

Figure A-6: Unstable Operation

Figures A-7 and A-8 were produced by changing the simulator so that each link was given the transmit rate of each packet as it was enqueued, rather than as it was transmitted. Otherwise, the parameters of the simulation are unchanged.

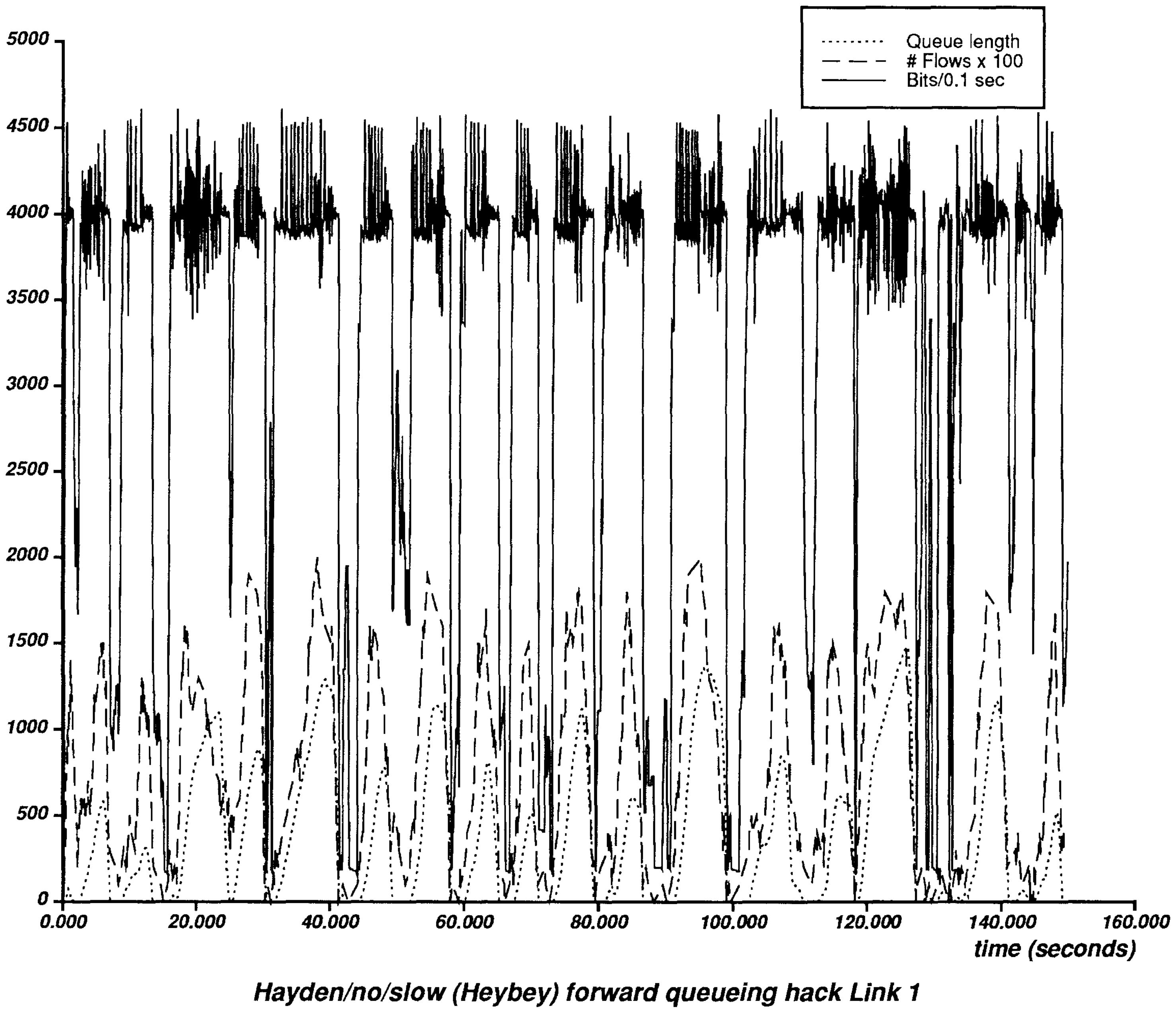

Figure A-7: Attempt to Solve Forward Queueing Problem 1

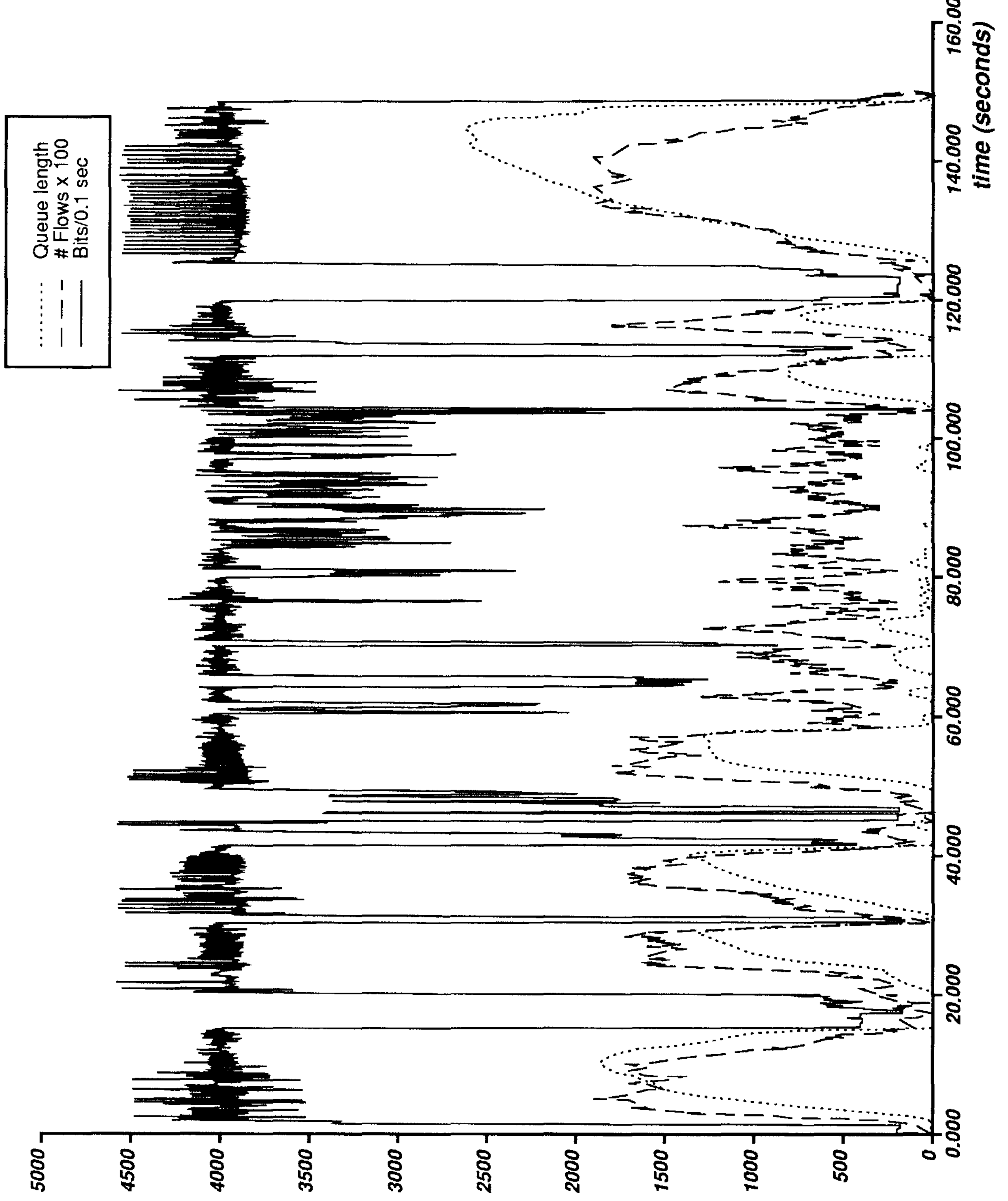

Figure A-8: Attempt to Solve Forward Queueing Problem 2

References

[Clark 87] David D. Clark, Mark L. Lambert, and Lixia Zhang.
NETBLT: A High Throughput Transport Protocol.
In J. J. Garcia-Luna-Aceves (editor), *Frontiers in Computer Communications Technology*, pages 353-359. ACM SIGCOMM, 1987.
Proceedings if the ACM SIGCOMM '87 Workshop.

[Hayden 81] H. Hayden.
Voice Flow Control in Integrated Packet Networks.
Master's thesis, Massachusetts Institute of Technology, 1981.
Department of Electrical Engineering and Computer Science.

[Jacobson 88] Jacobson, Van.
Congestion Avoidance and Control.
In John C. Burruss (editor), *Proceedings, SIGCOMM '88 Workshop*, pages 314--329. ACM SIGCOMM, ACM Press, August, 1988.
Stanford, California.

[Jaffe 80] J. M. Jaffe.
A Decentralized 'Optimal' Multiple-User Flow Control Algorithm.
In *ICCC Conference Record.* 1980.

[Lambert 87] Mark Lambert.
On Testing the NETBLT Protocol over Divers Networks.
Network Information Center RFC-1030, SRI International.
November, 1987

[Lambert 88] Mark Lambert.
May, 1988
Personal communication regarding a dynamic rate control algorithm for NETBLT.

[Mosely 84] Jeannine Mosely.
Asynchronous Distributed Flow Control Algorithms.
PhD thesis, Massachusetts Institute of Technology, October, 1984.
Department of Electrical Engineering and Computer Science.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TR470	5. MONITORING ORGANIZATION REPORT NUMBER(S) NO 0014-83-k-0215 NAG 2-582
6a. NAME OF PERFORMING ORGANIZATION MIT Lab for Computer Science 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Dept. of Navy
6c. ADDRESS *(City, State, and ZIP Code)* 545 Technology Square Cambridge, MA 02139	7b. ADDRESS *(City, State, and ZIP Code)* Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 1400 Wilson Blvd. Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Rate-Based Congestion Control in Networkd with Smart Links

12. PERSONAL AUTHOR(S)
Heybey, Andrew T.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Technical	FROM ______ TO ______	1990 January	36

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Networks, rate-based congestion control, network ocillation

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

I use a network simulator to explore rate-based congestion control in networks with "smart" links that can feed back information to tell senders to adjust their transmission rates. This method differs in a very important way from congestion control in which a congested network component just drops packets — the most commonly used method. It is clearly advantageous for the links in the network to communicate with the end users about the network capacity, rather than the users unilaterally picking a transmission rate. The components in the middle of the network, not the end users, have information about the capacity and traffic in the network.

I experiment with three different algorithms for calculating the control rate to feed back to the users. All of the algorithms exhibit problems in the form of large queues when simulated with a configuration modeling the dynamics of a packet-voice system. However, the problems are not with the algorithms themselves, but with the fact that feedback takes time. If the network steady-state utilization is low enough that it can absorb transients in the traffic through it, then the large queues disappear. If the users are modified to start sending slowly, to allow the network to adapt to a new flow without causing congestion, a greater portion of the network's bandwidth can be used.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL R. E. Schecter	22b. TELEPHONE *(Include Area Code)* (617) 253-5894 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

*U.S. Government Printing Office: 1985-507-047

Unclassified