TCP/IP and HIPPI Performance in the CASA Gigabit Testbed *

Bilal Chinoy
San Diego Supercomputer Center
bac@sdsc.edu

Kevin Fall
San Diego Supercomputer Center †
kfall@cs.ucsd.edu

Abstract

We investigate the packet delay and loss characteristics of the wide-area HIPPI-based CASA gigabit testbed. Developed for high–speed local area device interconnects, HIPPI is a point–to–point, connection–oriented protocol. We show HIPPI blocking can degrade performance by increasing delay and/or packet loss. In the CASA network under conditions of blocking, a tradeoff exists between packet loss and delay variance. The tradeoff point is determined by a combination of factors: source packet rate, mean blocking rate, and a *configurable* connection establishment timeout threshold. We show the delay/loss tradeoff manifests itself in TCP as inducing either the *slow-start* algorithm or requiring TCP to adjust retransmission timeout values due to increased delay variance. In the former case, we have measured TCP throughput to drop by as much as 97% of its unblocked maximum, as compared with 75% for the latter case.

1 Introduction

The desire to interconnect high–speed peripherals and supercomputers has lead to the development of several link–layer protocols offering bandwidths above 100MBytes/s. Among these, the High Performance Parallel Interface (HIPPI) [6], as specified by the ANSI X3T9 Standards Committee, has enjoyed great popularity due to the availability of HIPPI interfaces for most supercomputers and high–speed peripheral equipment. HIPPI enables computers and peripheral devices to communicate at bandwidths of 800(1600) Mb/s over a 64(128) wire parallel cable, with a maximum distance of 25 m.

In an effort to provide a prototype distributed supercomputer computation environment, the CASA gigabit testbed was formed [5]. At present, CASA links geographically distributed HIPPI networks some 100 miles distant using gateways capable of encapsulating HIPPI data over fiber-optic SONET[12] links at 800Mb/s. In this paper, we describe the delay and loss characteristics of HIPPI networks as extended to a wide area in the CASA network. In section 2 we describe the motivation for examining the HIPPI protocol dynamics in the CASA environment. We then give a brief discussion of the operation of a conventional HIPPI network in section 3. In section 4 we describe the CASA network and how it differs from a conventional HIPPI local–area network. We then develop a model of port contention or *blocking* in section 3.2, and evaluate the effects of blocking in sections 6 and 7. Section 8 concludes.

*Supported by National Science Foundation and the Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives.

†Kevin Fall is also a member of the Computer Systems Laboratory, University of California, San Diego

2 Motivation

An important goal for the CASA effort is to understand key factors contributing to the performance of applications distributed across multiple supercomputers. Applications are divided into several *subproblems*, and executed on architectures most appropriate to the computational needs of each subproblem[1]

Supercomputer applications divided into subproblems may overlap computation with network communication [8]. The effect of network latency can be completely hidden if subproblem execution times are carefully matched to network speed. Achieving this *latency hiding* is critical to the performance of distributed applications.

Blocking may occur because HIPPI is a connection–oriented point-to-point protocol allowing no simultaneous connections. Blocking implies data may not be delivered to a destination because of a competing (unrelated) concurrent connection. By affecting delay and packet loss, synchronization between application subproblems may be lost, eventually affecting overall application performance as measured in total wall-clock time. Our goal in this study is to understand and quantify the effect of HIPPI blocking on the delay and loss of TCP/IP traffic.

3 HIPPI Overview

A HIPPI local area network typically consists of a crossbar switch with nodes attached in a star configuration. Each switch attachment point provides a distinct *input* and *output* port (64-wire cable). Connections may be initiated by an attached HIPPI node through a switch's input port to a specified output port. Connections are simplex in all cases, and may be set up and torn down rapidly. The switch itself is a passive device; it does not introduce delay after a connection has been established between two nodes.

3.1 HIPPI Protocol Description

The HIPPI protocol is *simplex* and *connection oriented*. Communicating HIPPI nodes must explicitly set up, manage, and tear down a connection. Bidirectional communication requires each node to act as both a source and destination. A HIPPI source may request a connection to a destination by raising the *REQUEST* electrical signal while simultaneously placing *connection control information* (*CCI*), including the address of the destination, on the data lines. The cross–point switch decodes the *CCI*, determines the correct output port, and attempts to connect the source and destination ports. If the destination port is in use by another active connection or the destination node refuses connections, the source receives a connection rejected message and no connection is established.

If a connection is established, the destination node acknowledges and accepts the connection by raising the *CONNECT* electrical signal. Either side may terminate the current connection at any time by deasserting either the *REQUEST* signals (at the source) or *CONNECT* (at the destination). Connections otherwise remain open indefinitely. Generally, a connection set-up requires a round trip time (RTT) delay, amounting to only about half a microsecond[2] when cable distances are limited to 25 m. Once a connection has been established, the switch is not directly involved in data delivery.

HIPPI delivers data in arbitrarily long *frames*. Frames are divided into a sequence of "full–sized" *bursts* and possibly a single "short" burst. Full–sized bursts contain 256 32-bit words of data each. Once a connection has been established, bursts of data are sent from the source port to the destination port, subject to flow control. The *READY* signal provides flow–control signaling from a HIPPI destination to its peer source. No round trip cable delays are incurred in the sending of bursts. A destination node may generate up to 63 *look–ahead* READY signals avoiding throughput degradation experienced by stop–and–wait operation.

[1] For example, a parallel scalar computation such as a lattice-gauge problem may be best implemented on a massively parallel machine and a floating point matrix multiply may be best executed on a fast vector machine.

[2] This value represents round trip delay and hardware overhead only.

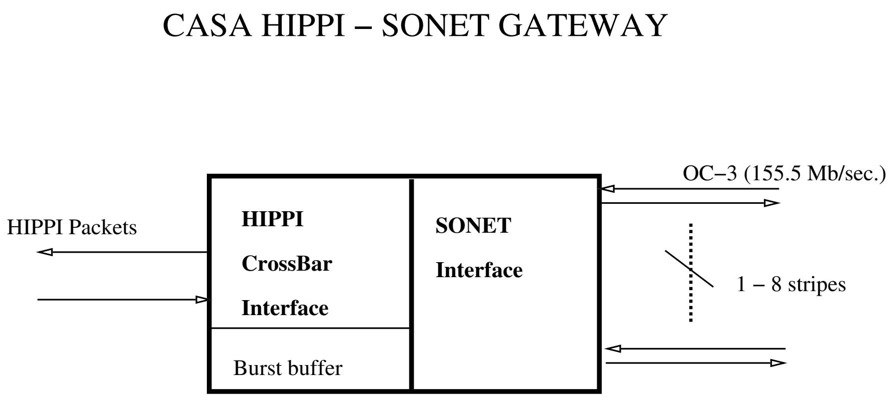

Figure 1: HIPPI–SONET Gateway

The *HIPPI–SONET* gateway extends HIPPI LANs across wide areas using SONET OC-3 links. It provides *local termination* of HIPPI connections, avoiding performance degradation caused by long round–trip delays. It includes 4MB of *burst buffers* to accommodate wide–area bandwidth-delay products.

3.2 Blocking Behavior

When a HIPPI destination has an active connection, no other connections to it can be established. An arriving request destined for a busy destination is rejected. Failure to establish a connection to a busy destination is called *blocking*; the requesting source is said to be *blocked*. The HIPPI protocol allows for arbitrarily long data packets, implying connections may be active indefinitely. In practice, packets are of finite length due to application or network layer framing. In the case of IP [2], the maximum size of the datagram is limited to 64 KB, which requires a 640 microsecond transmission time on a 800 Mb/s HIPPI network. However, an application using HIPPI directly (*raw* HIPPI) may send much larger packets, resulting in a correspondingly longer connection time. Applications displaying graphics output to a HIPPI–attached frame buffer are typical of the latter category.

A blocked source may retry to access a busy destination. The retry mechanism is configurable on a per–connection basis. With the *distributed* connection management policy, the source itself retries immediately or after some (usually random) time to establish a new connection. In the *centralized* policy, the switch itself arbitrates between competing connection requests to a single destination.

The HIPPI protocol specification supports the centralized policy by having sources set the *camp-on* (CO) bit in the connection request. Presence of the CO bit in a connection request signals the switch to cache the request and service it as soon as the requested destination becomes free. Camp-on is accomplished electrically by asserting the source *CONNECT* signal. The HIPPI–SC specification [6] does not specify a particular arbitration algorithm to be used when multiple sources are camped on a single busy destination.

4 The CASA Gigabit Testbed

The principal objective of the CASA gigabit testbed project is to build a distributed heterogeneous supercomputing environment supporting execution of large-scale scientific applications [5]. Because many supercomputers already have HIPPI interfaces and are part of local HIPPI networks, the CASA network was designed to interconnect geographically distant HIPPI networks using available fiber-optic technology.

A special purpose device called a *HIPPI-SONET* gateway (HSG) [9], has been designed and built by the Los

Alamos National Laboratory, for the CASA project, and is illustrated in Figure 1. An HSG is connected to a local HIPPI crossbar-based network and accepts packets destined for remote HIPPI nodes from local nodes. Incoming HIPPI data from the LAN is inserted into the *payload* portion of a SONET stream. A companion gateway at the remote end of the SONET link extracts the payload portion from the SONET stream, repackages the data as HIPPI frames and tries to deliver the frames to attached destination node.

The CASA wide-area link must be of at least 800 Mb/s bandwidth to ensure full rate HIPPI interconnectivity. Current SONET interface technology is not mature enough to provide a single fiber circuit at HIPPI speeds. Consequently, the CASA approach is to *stripe* HIPPI data across multiple circuits. The HSG gateways can stripe across up to 8 OC-3 links[3] each providing 155.5 Mb/s bandwidth.

4.1 Flow Control and Buffer Management

A destination HIPPI node asserts the *READY* signal, permitting the source to transmit data. The simple stop–and–wait nature of this flow control protocol is effective only because of small signal latencies. If READY signals are delayed while traversing long distances, the stop–and–wait protocol would be unable to fully utilize the available capacity. The HSG provides *local termination* of wide-area HIPPI connections, eliminating the need for end–to–end *READY* signaling while maintaining end-to-end transport of HIPPI frames.

A HIPPI conversation between remote machines across CASA is actually composed of 3 separate connections. In order to transmit data, the source device sets up a connection with the local HSG. Data flows across a *pseudo-HIPPI* connection between the pair of HSGs. No HIPPI signals are passed over a pseudo-HIPPI connection. Data arriving at the remote end is delivered by the destination HSG via a conventional HIPPI connection to the destination node.

The local HSG generates *READY* signals to the source node and sends data to the remote HSG across the SONET link. The remote HSG must have enough buffer capacity to queue received data while it attempts to establish a connection to the destination node.

Two interesting queueing scenarios arise in the long-distance CASA network:

- When data arrives at a remote HSG faster than it can be delivered to destination nodes, the remote HSG stores data in *burst buffers.* A local HSG flow-controls a source node whenever the remote HSG signals burst buffer saturation. Local HSGs may flow-control a source by either throttling the generation of *READY* signals or refusing new connections. Flow-control causes delays while packets wait in device driver and hardware queues.
- When a remote HSG attempts to establish a HIPPI connection to a destination and fails, it buffers any packets associated with the pending connection and retries. Packets arriving during this *retry* period experience *head–of–line* queueing. The *gateway hold parameter* h defines the HSG retry period. All queued packets associated with the pending connection are discarded if the hold parameter timer expires. The hold parameter may be set to ∞, implying the HSG retries forever.

In the following sections we explore the consequences of these scenarios.

5 A HIPPI Blocking Model

HIPPI destination blocking occurs during periods of contention for a node's input port. We call an end–to–end conversation being degraded by HIPPI blocking the *blocked* connection. The sender on the blocked connection transmits packets at a deterministic rate λ. For our environment, there is a one–to–one correspondence between

[3] The OC designation in the SONET standard stands for Optical Carrier. The basic multiplexing building block for SONET is OC-1, which is 51.84 Mb/s.

packets and HIPPI connections because all nodes agree by convention to break connections between packets. We shall call a simplex connection competing with the blocked connection the *blocking* connection. The blocking connection keeps the blocked connection's receive port busy for a set of time intervals given by an exponential distribution with parameter μ. For a mathematical model of HIPPI switches, the reader is referred to [4].

The packet (connection) interarrival rate is deterministic with parameter λ, and the service rate is equivalent to the blocking connection interarrival rate μ. The CASA network will operate in one of two modes when $\lambda > \mu$, based on the configurable parameter h. For $h = \infty$, the HSG will only discard packets when burst buffer space is exhausted. When $0 < h < \infty$ (finite h), the HSG will attempt to deliver a HIPPI packet to the proper destination for at most $20h$ microseconds.[4] In summary, when $\lambda > \mu$ and $h = \infty$, buffer occupancy will grow (and thus introduce a larger mean packet delay) in proportion to λ until packets are discarded. Alternatively, when $\lambda > \mu$ and h is finite, buffer occupancy will remain bounded in proportion to h but packet discards will increase in proportion to μ.

We now turn to the expected effects of HIPPI blocking on TCP. We first review TCP's congestion control and retransmission timer setting policies. A complete description of the TCP congestion control scheme is introduced by Jacobson in [1]. A simulation study of the congestion control scheme with multiple TCP connections is presented in [10].

In TCP, packet loss is interpreted as an indication of network congestion. TCP assumes packet loss has occurred when either a retransmission timer has expired, or a number of duplicate acknowledgements have been received. In the first case, *slow–start* is initiated to reduce network load quickly, followed by a recovery period to re-establish a point of equilibrium in which packet injection rate matches packet removal rate. In the latter case, *congestion avoidance* reduces the sending rate, but not as drastically as slow-start. The TCP congestion control algorithms are realized by introducing a congestion window *cwnd* in the sender in addition to the normally–maintained window advertised by the receiver for flow control (*advertised_window*). The current send window is always set as $\mathsf{MIN}(advertised\_window, cwnd)$.

Slow–start encompasses two techniques. It reduces *cwnd* to a single segment after a retransmission timer has expired. To recover, it includes an *additive increase* policy in which the send window is increased by one segment for each ACK received. Thus, slow–start grows the congestion window exponentially with respect to time, assuming no segments are lost. Slow–start operation is maintained at its exponential growth rate until the *cwnd* reaches the "slow start threshold" *ssthresh*. After *ssthresh* has been reached, the congestion window is grown linearly by incrementing it each time a window's worth of ACKs is received. In the event of retransmission timeout expiration, *ssthresh* is set to one half the previous window.

TCP enters *congestion avoidance* when a small number[5] of duplicate acknowledgements for the same segment have been received. Here, the current window is reduced by half, and *ssthresh* is reassigned to this new value (not to 1, which was the case for slow–start).

The above algorithms are actually independent and serve distinct functions. The slow–start algorithm attempts to "ramp–up" a sender's rate from zero until the available channel capacity is utilized. Slow–start is run only after a retransmission timer has expired (or a new connection is being initiated) when the channel is devoid of traffic. Congestion avoidance is run in all other cases, usually when the channel contains packets in transit. Its purpose is to track fluctuations in available channel bandwidth and modulate send rate slowly.

TCP sets segment retransmission timers according to an estimator taking both RTT mean and variation into account. When retransmissions do occur, subsequent timeout values are assigned exponentially. Estimating the RTT variance allows TCP to adapt its retransmission timer appropriately to a system with a large variation in RTT (especially useful in a multiaccess wide area network). The exponential backoff helps to avoid excessive retransmissions in lossy networks.

HIPPI blocking as described above can affect TCP by either blocking data segments (*segment blocking*) or ACKs

[4] The value $20h$ represents the amount of wallclock time required for a hardware countdown timer to reach zero if initialized with value h.

[5] This value, *tcprexmtthresh*, is typically set to 3.

(*ACK blocking*). We pursue the following intuitive description:

When segments are delayed but not discarded ($h = \infty$), both RTT mean and RTT variance increase. Acknowledgements are still delivered quickly, so the ACK return rate will closely reflect the trouble the data segments are experiencing, resulting in an accurate "ACK clock" . Spurious retransmissions should thus be limited. For a finite h, segments may be discarded, resulting in retransmission timers expiring (which will induce slow-start behavior and thus reduce throughput). When ACK blocking occurs, ACKs experience increased delay mean and variance when $h = \infty$ but high loss and bounded delay when h is finite.

In effect, segment or ACK loss should affect a sending TCP similarly depending on the parameter h. We expect buffer overrun to be less frequent in ACK blocking due to the smaller packet size of ACKs during a unidirectional TCP data flow. These expectations are evaluated in the next sections.

CASA EXPERIMENTAL ENVIRONMENT

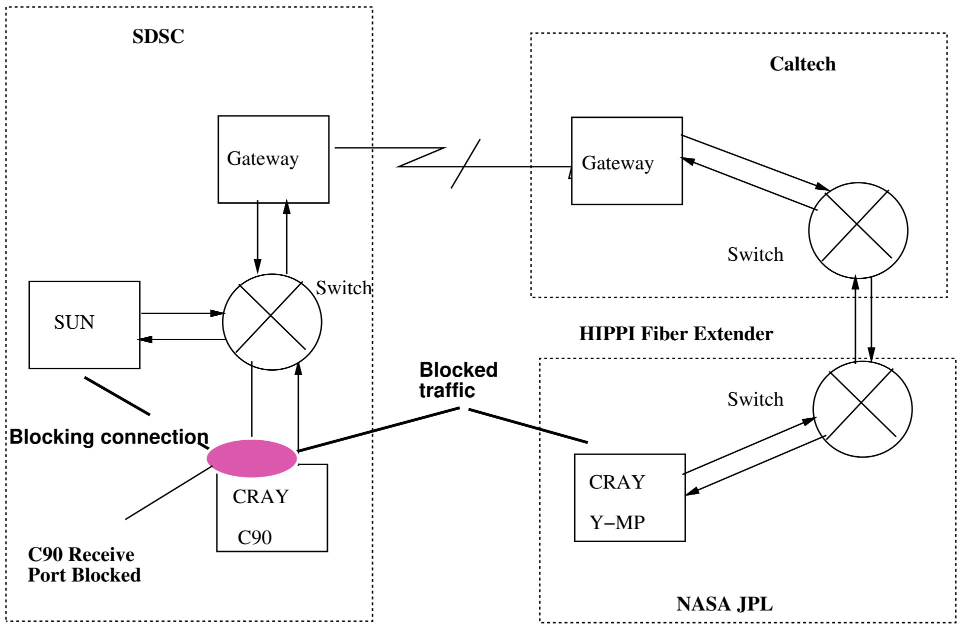

Figure 2: Experimental Configuration

A HIPPI wide-area connection (*blocked* connection) is established between the SDSC Cray C90 and JPL Cray YMP. A Sun workstation attached to the SDSC HIPPI LAN acts as a *blocking* connection source.

6 Experimental Environment

Our experimental configuration is shown in Figure 2. At the SDSC site, we have a Cray C90 and a Sun workstation on the HIPPI LAN based on the NSC PS–32 HIPPI crossbar switch [7]. We used 4 out of the 8 OC-3 links between SDSC and CalTech, providing an aggregate bandwidth of approximately 620 Mb/s.[6]

The CalTech site has two supercomputers connected to the HIPPI LAN, but neither of these are currently capable of using the TCP/IP protocol suite. CalTech is linked to JPL via a HIPPI fiber extender.[7] The extenders provide

[6] We were limited to 4 stripes when the experiments were performed.

[7] A HIPPI fiber extender is a device capable of transmitting all the HIPPI electrical signals across a fiber-optic link. The use of this

transparent access between CalTech and JPL.

We create a blocked connection between the JPL Y-MP and the SDSC C90. On the blocked connection, we vary both the packet size and the packet source rate. We choose two packet sizes: 64bytes and 64 Kbytes, representing typical cases of interactive and bulk data transfer, respectively. We choose three packet rates: 1 pps, 10 pps and 100 pps. Round trip packet delay and loss were measured using the *ping* program on the SDSC C90.

The Sun workstation is used to create the blocking connection by establishing HIPPI connections to the SDSC C90 for a random time period. The blocking connection is modeled by selecting the blocking time as an exponentially distributed random variable with parameter μ. We term the mean blocking time *blocking factor* ($BF = 1/\mu$), and vary it between 0.001 sec. and 1 sec. We choose two values for the gateway hold time h: ∞ and 10 ms. For the TCP blocking study, we establish a bulk-data transfer with ACK blocking between the JPL Y-MP and the SDSC C90. The *File Transfer Protocol* (FTP) [11] is used to transfer data. The window scaling option as specified by RFC1323 [3] is set to 4, allowing the receiver to advertise a window up to 1048560 bytes. The receiver socket buffer size was set to 400 Kbytes, and 41 Mbytes were transferred in each test. TCP traces are collected using the *trcollect* utility, provided on Cray systems.

We present our measurements in the next section, along with a discussion of the implications of each set of observations.

7 Analysis

In this section we examine the effect of blocking factor and gateway hold time on packet loss and delay using an unreliable protocol (ICMP/IP). We then turn to the effects of HIPPI blocking on a TCP/IP connection.

7.1 Gateway hold time $h = \infty$

Figure 3 shows the mean packet **delay** on the blocked connection as a function of blocking factor, for several source packet rates and sizes. When $\lambda < \mu$, mean delay increases with blocking factor. When $h = \infty$, a remote HSG never discards packets but instead allows queues to build. The mean delay thus increases with the blocking factor. Furthermore, mean delay is insensitive to packet size because per packet connection wait time dominates the per-byte transmission time. When $\lambda > \mu$, mean delay grows unbounded (ultimately resulting in source HSG flow control), as indicated by the vertical arrows. For example, in the 100 pps cases, this occurs at a blocking factor of 0.01.

Figure 4 shows the measured **packet loss** percentage corresponding to the delay graph Figure 3. When $h = \infty$, packet loss only occurs when the HSG burst buffers overflow. Observed packet loss is due to packet discards at a hardware flow–controlled sender which does not employ higher–layer flow control (in our case ICMP/IP as used by *ping*). When $h = \infty$, an HSG with full burst buffers will signal the remote HSG to stop accepting new connections. Loss remains constant at 0 except when, for a fixed blocking factor, the source $rate * size$ product is large enough to overflow the burst buffers (the *saturation point*). For example, in the 64 Kbyte-100 pps case, the saturation point occurs for $0.005 \leq BF < 0.01$. Beyond the saturation point (provided $\lambda < \mu$), packet loss grows exponentially with the blocking factor for a fixed $rate * size$ product.

7.2 Gateway hold parameter $h = 10ms$

Figure 5 recreates the tests performed in Section 7.1 but with the gateway hold parameter $h = 10ms$. In this case, an HSG unable to establish a connection in 10ms will discard queued traffic for the pending connection. Thus,

scheme is limited to approximately 10 km distance, beyond which HIPPI signal regenerators are required. The fiber-extender is scheduled to be replaced with a pair of HSGs.

the mean RTT of a blocked (but successfully delivered) packet is bounded by $h + (unblocked)RTT$. This effect is illustrated by the increase in mean delay for $BF < h$. When $BF \geq h$, we observe a horizontal trend toward the $h + (unblocked)RTT$ bound. In our case, the mean unblocked RTT ranges between 4ms and 6ms for packets sized 64bytes and 64Kbytes, respectively.

Packet loss for the $h = 10ms$ experiment is illustrated in Figure 6. We observe loss to be largely insensitive to both packet size and rate for a fixed blocking factor. When $BF < h$ (below 0.01), packet discards occur exactly when the instantaneous blocking time exceeds h. Because the blocking connection's hold time is exponentially distributed, its standard deviation is equal to its mean, and thus aggregate packet loss increases with proximity to h. For a fixed BF, each instance of instantaneous blocking time exceeding h causes packet loss. In comparison to the $h = \infty$ case, this case results in a higher aggregate packet loss. When $BF > h$, only those packets encountering an instantaneous blocking time below h will be successfully delivered. As BF increases the probability of successful delivery decreases until no packets can be delivered.

7.3 TCP

We now turn our attention to the effect of packet loss and delay on TCP in our experimental environment. Figures 7 and 8 show the sequence number and congestion window versus time for three distinct TCP conversations. Without blocking (and without the 100pps limitation from sections 7.1 and 7.2), we measured an upper bound on application throughput of 314.4Mb/s.

For the blocked cases, BF is fixed at 0.02 seconds for the two values of h used previously. The blocking connection, initiated by the Sun workstation, keeps the receive port of the C90 busy. During a unidirectional file transfer from the C90 to the YMP, ACK packets returning from the YMP to the C90 may be lost or delayed.

When $h = \infty$, ACKs are delayed but not lost, causing throughput degradation. The measured throughput in this case is 76.8 Mb/s, only 24.4% of the best case. The sender's congestion window ramps up to the receiver's advertised window and remains unaffected by the delayed ACKs, as illustrated in Figure 8.

When $h = 10ms$, the greater ACK loss rate causes the sender's retransmission timer to expire, inducing slow-start behavior. The measured throughput in this case is 10.4 Mb/s, or 3.3% of the best case. This is illustrated in Figure 7 by observing that segments are retransmitted and in Figure 8 by the sawtooth character of the congestion window plot, in contrast to the congestion window seen in the no-blocking and $h = \infty$ cases.

8 Conclusions

In this study, we have verified our hypothesis that HIPPI blocking behavior occurs and has a measurable effect on packet loss and delay. We have also observed the measured character of packet loss and delay is associated with the gateway hold time parameter h as expected. In the presence of blocking, there exists a tradeoff between larger round–trip delays (and variance) versus greater packet loss. For infinite h, packet loss is reduced at the expense of increased delay (and variance), in contrast with finite h which provides bounded delays and queue occupancies at the expense of increased packet loss.

When $h = \infty$, the CASA HIPPI WAN resembles a HIPPI LAN. In a local HIPPI environment, the switching fabric will not discard frames because of congestion. Similarly, in the CASA network when $h = \infty$, flow control preserves local HIPPI reliability semantics by instituting admission control during saturation periods (in preference to dropping packets). We therefore anticipate all CASA findings with $h = \infty$ to apply as well to HIPPI LANs.

The delay–vs–loss tradeoff affords the opportunity to match network characteristics with application/transport requirements. With respect to TCP, high packet loss induces the slow-start policy even when the network may not be congested, leading to low throughput. In addition, the variance-sensitive TCP RTT estimator is sufficiently robust to avoid spurious retransmissions in HIPPI–based networks such as CASA. Thus, the delay/loss tradeoff for TCP clearly favors limiting packet loss. Other transport protocols or loss–tolerant applications may prefer a less reliable channel in exchange for bounded delay variance.

In a general purpose network, applications should be permitted to specify a preference with respect to the tradeoff mentioned above by indicating a desired type of service. The HSG presently lacks support for setting h based on type-of-service. Furthermore, supporting type of service requires protocol-level identification of an application's preferences, and is not supported by HIPPI. Providing router functionality within the HSG and employing a network layer protocol capable of specifying type of service handling would allow HIPPI LANs to be effectively extended to the WAN environment.

References

[1] Jacobson, V., "Congestion Avoidance and Control", Proc. ACM SIGCOMM 1988, Stanford, CA, Aug 1988.

[2] Postel, J., "Internet Protocol", RFC 791, Sep 1981.

[3] Jacobson, V., Braden, R., Borman, D., "TCP Extensions for High Performance", RFC 1323, May 1992.

[4] Chlamtac, I., Ganz, A. and Kienzle, M., "An HIPPI Interconnection System", IEEE Trans. Comput., vol 42, pp. 138–149.

[5] P. Messina, "CASA Gigabit Network Testbed", Proc. Supercomputing '91, November 1991

[6] ANSI X3T9.3, "High-Performance Parallel Interface:HIPPI-PH, HIPPI-SC, HIPPI-FP and HIPPI-LE", Amer. Nat'l. Std. for Info. Sys., Aug 1993.

[7] Network Systems Corporation, *HIPPI Connectivity Solutions*, 1993

[8] Moore, R. "Distributing Applications Across Wide Area Networks", General Atomics Technical Report GA-A20074, May 1990.

[9] St. John, W., *Personal Communication*, Feb. 1994.

[10] Shenker, S., Zhang, L., "Some Observations on the Dynamics of a Congestion Control Algorithm", Computer Communications Review, Vol. 20, no. 5, Oct. 1990.

[11] J. Postel, J. Reynolds, "File Transfer Protocol", RFC 959, Jan 1985.

[12] Ballart, R., Ching Y., "SONET: Now it's the Standard Optical Network", IEEE Communications Magazine, Vol. 29, no. 3, Mar. 1989.

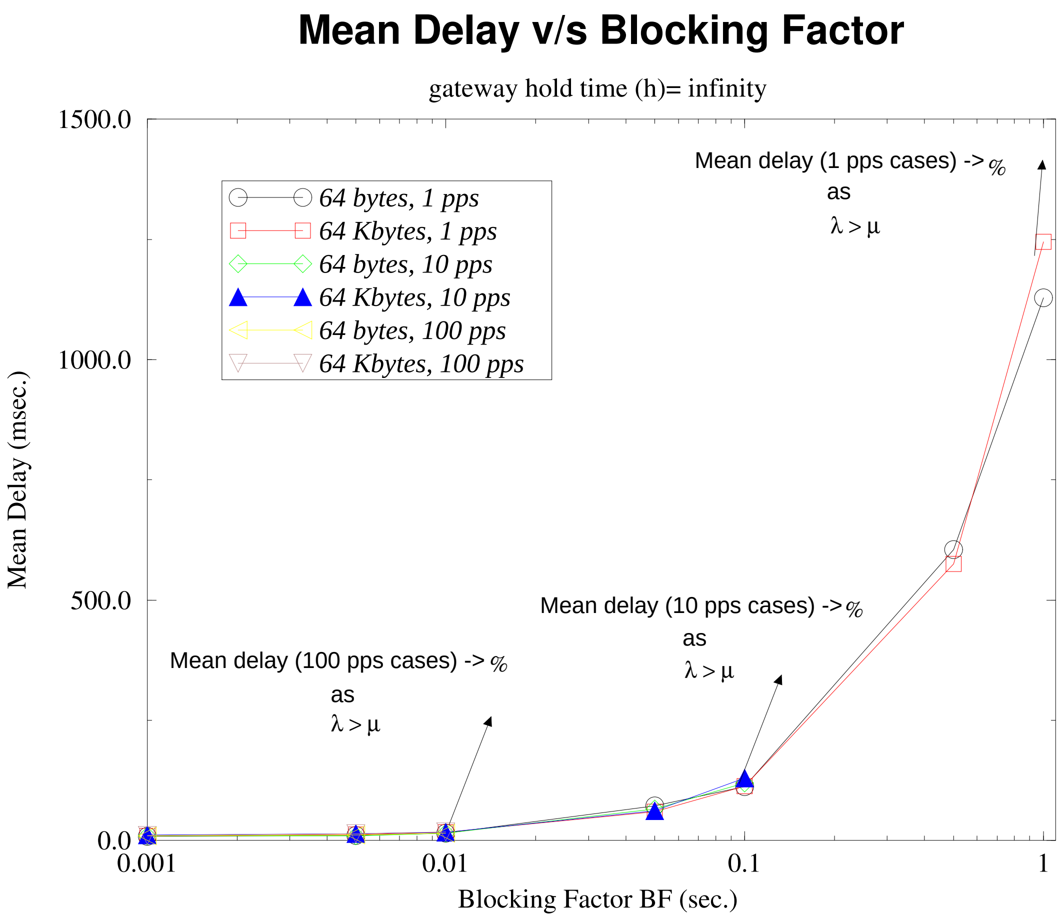

Figure 3: Mean Packet Delay vs Blocking Factor for $h = \infty$

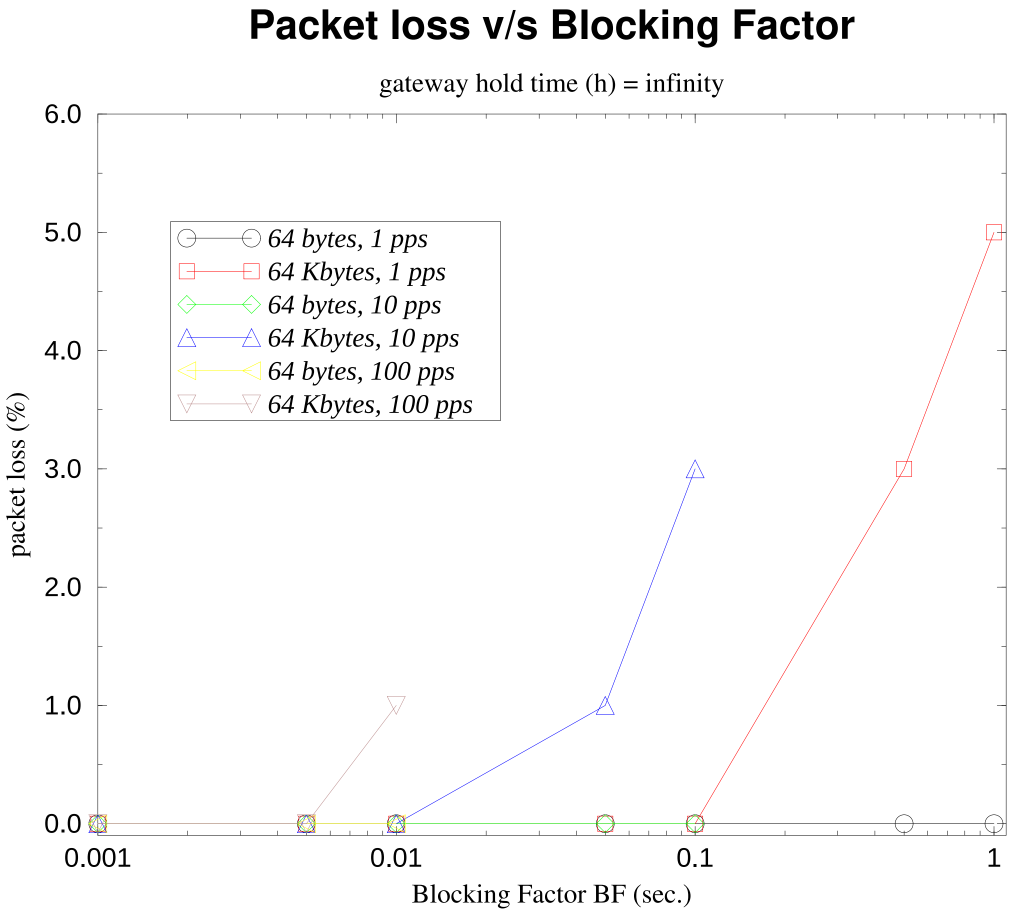

Figure 4: Packet Loss vs Blocking Factor for $h = \infty$

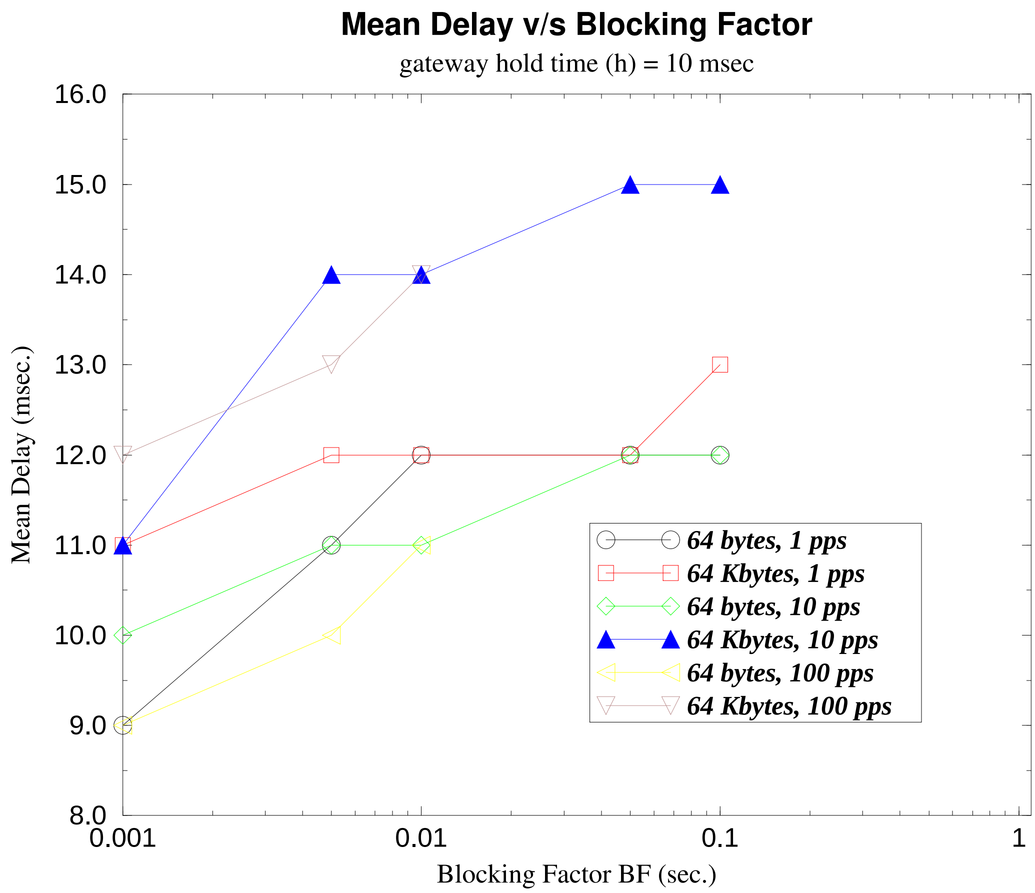

Figure 5: Mean Packet Delay vs Blocking Factor for $h = 10ms$

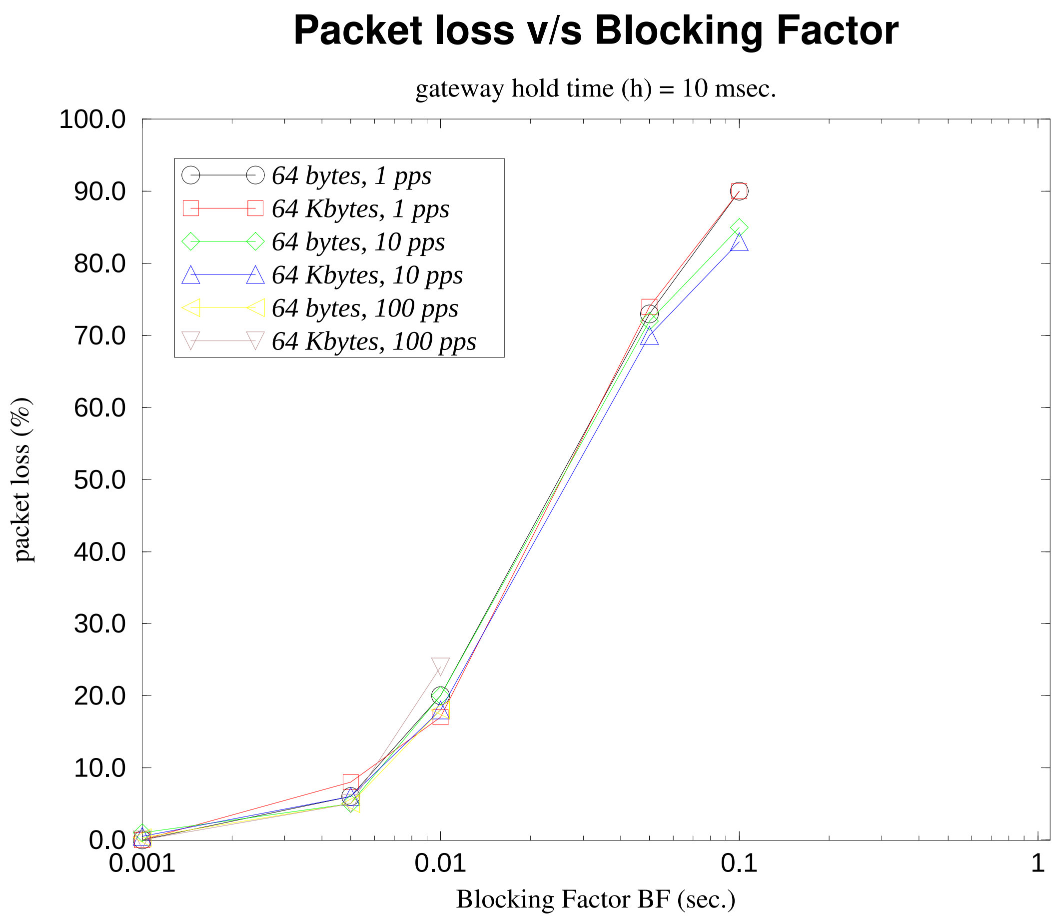

Figure 6: Packet Loss vs Blocking Factor for $h = 10ms$

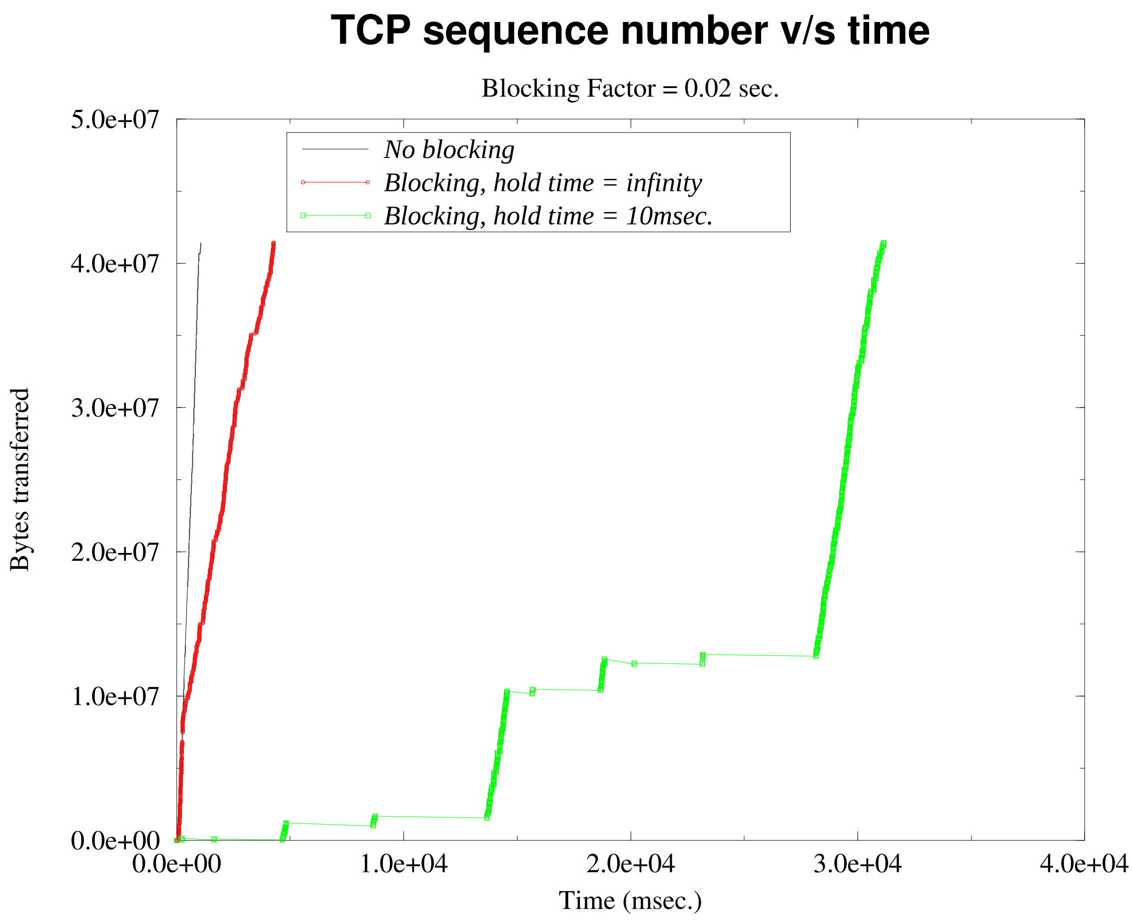

Figure 7: TCP Sequence Number vs Time

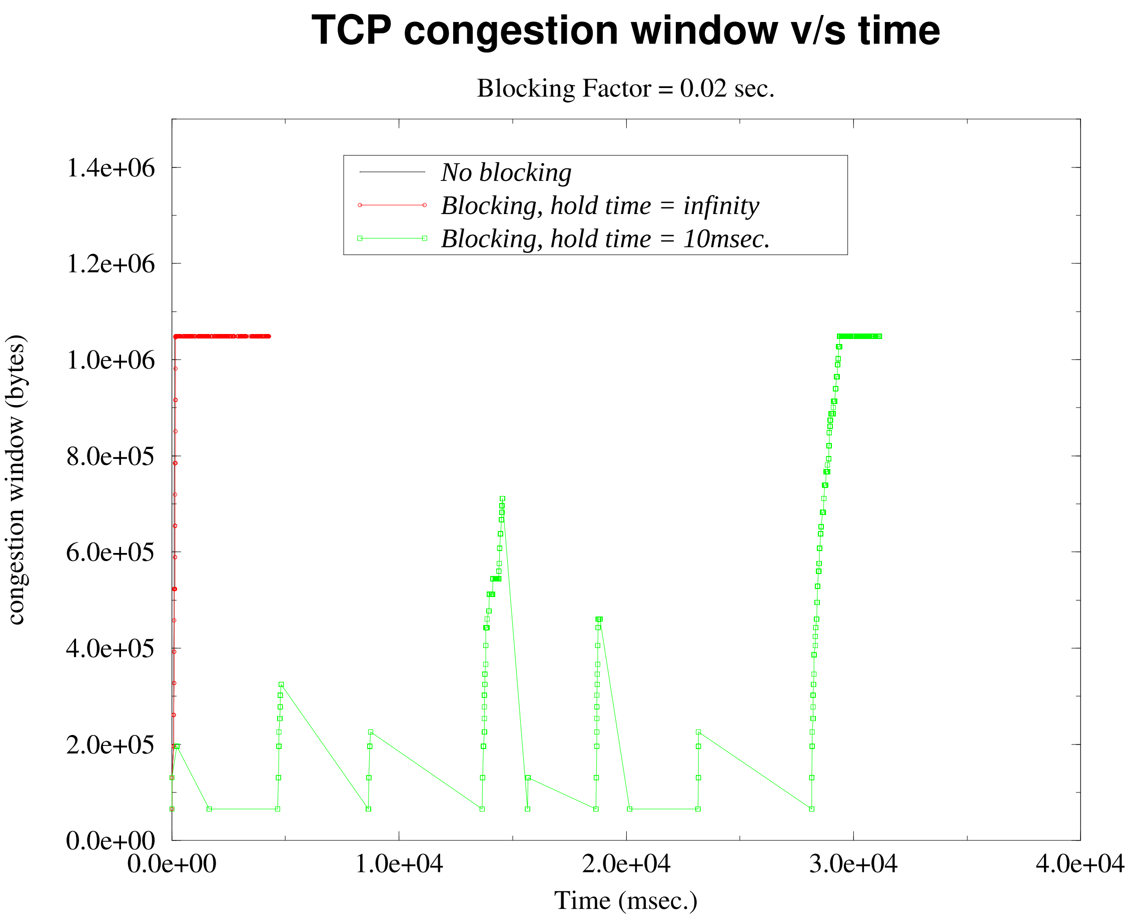

Figure 8: TCP Congestion Window vs Time