# TCP for High Performance in Hybrid Fiber Coaxial Broad-Band Access Networks

Reuven Cohen, *Member, IEEE*, and Srinivas Ramanathan

*Abstract*—**Motivated by the phenomenal growth of the Internet in recent years, a number of cable operators are in the process of upgrading their cable networks to offer data services to residential subscribers, providing them direct access to a variety of community content as well as to the Internet. Using cable modems that implement sophisticated modulation–demodulation circuitry, these services promise to offer a several hundredfold increase in access speeds to the home compared to conventional telephone modems. Initial experiences indicate that cable networks are susceptible to a variety of radio-frequency (RF) impairments that can result in significant packet loss during data communication. In the face of such losses, the transmission control protocol (TCP) that is predominantly used by data applications degrades dramatically in performance. Consequently, subscribers of broad-band data services may not perceive the projected hundredfold increase in performance. In this paper, we analyze the performance of TCP under different network conditions using simulations and propose simple modifications that can offer up to threefold increase in performance in access networks that are prone to losses. These modifications require only minor changes to TCP implementations at the local network servers alone (and not at subscribers' PC's).**

*Index Terms*— **Broad-band access, hybrid fiber coaxial networks, residential data services, TCP performance.**

## I. Introduction

THE RECENT phenomenal growth of the Internet has opened up a vast market for high-speed data services to the home. To pursue this emerging market, a number of telephone carriers and cable operators are actively deploying various broad-band access technologies including wireline technologies such as asymmetric digital subscriber line (ADSL) over telephone copper lines, hybrid fiber coaxial (HFC) technology—a variant of today's cable networks, and fiber to the curb—an extension of the fiber in the loop concept [5]. Local multipoint distribution alternatives using wireless technologies are also under development. In this paper, we focus on broad-band data services offered over HFC access networks, which have emerged as a cost-effective technology for many cable operators and a few telephone carriers.

Using cable modems that employ efficient data modulation schemes, these HFC access networks are capable of transporting tens of megabits of information per second, thereby offering a several hundredfold increase in access bandwidth compared to conventional telephone modems [5]. However, initial experiences indicate that real-world HFC networks are susceptible to a variety of radio-frequency (RF) impairments that can result in significant packet loss during data communication [7]. In the face of such losses, the transmission control protocol (TCP), which is predominantly used by data applications, degrades dramatically in performance. Consequently, subscribers of broad-band data services may not perceive the projected hundredfold increase in performance.

This paper analyzes the performance of TCP under different HFC network conditions. Based on this analysis, we highlight architectural considerations and maintenance targets for HFC networks supporting data services. To enhance the performance of TCP in an HFC network during periods when the network is error prone, various methods of tuning TCP parameters are proposed. Simulation studies demonstrate that these methods are complementary to one another and can result in an over threefold increase in performance under certain loss conditions. A major attractiveness of these enhancements is that the performance improvements can be obtained by tuning TCP implementations at the HFC network servers alone, without requiring any changes in subscribers' PC's.

The rest of this paper is organized as follows. Section II outlines the typical architecture of a broad-band data system. Section III highlights the performance problems experienced by TCP applications in HFC networks. Section IV characterizes the performance of TCP under different network conditions. Various methods for enhancing TCP performance over HFC networks are discussed in Sections V and VI. Section VII discusses the implications of the TCP performance results for cable operators deploying broad-band data services.

## II. Broad-Band Data Service Architecture

Fig. 1 depicts a typical architecture of a broad-band data system that services residential subscribers in a metropolitan area. At the heart of this system is a local *server complex* that houses servers supporting a variety of community services including bulletin boards, newsgroups, electronic mail, directory services, Web access, etc., as well as caching servers to maintain local copies of Web pages that are frequently accessed from the Internet. The servers are interconnected by a high-speed asynchronous transfer mode (ATM) network. Routers and firewalls enable connectivity from the HFC network to external networks including the Internet. Data retrieved from the server complex is routed over the HFC network via one or more signal conversion systems (SCS's). To enable data transmissions to coexist with transmission of analog television signals, data transmissions over the HFC network are analog modulated. Frequency-division multiplexing is used over the

Manuscript received July 10, 1997; revised September 22, 1997; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Crowcroft.

R. Cohen was with Hewlett-Packard Laboratories, Haifa 32000, Israel. He is now with the Department of Computer Science, Technion, Haifa 32000, Israel (e-mail: rcohen@cs.technion.ac.il).

S. Ramanathan is with Hewlett-Packard Laboratories, Palo Alto, CA 94304 USA (e-mail: srinivas@hpl.hp.com).

Publisher Item Identifier S 1063-6692(98)01971-2.

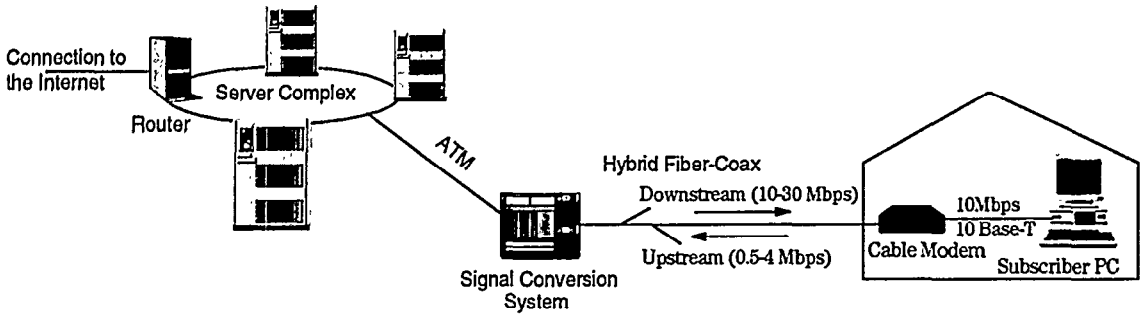



Fig. 1. Configuration for providing broad-band data services over an HFC network.

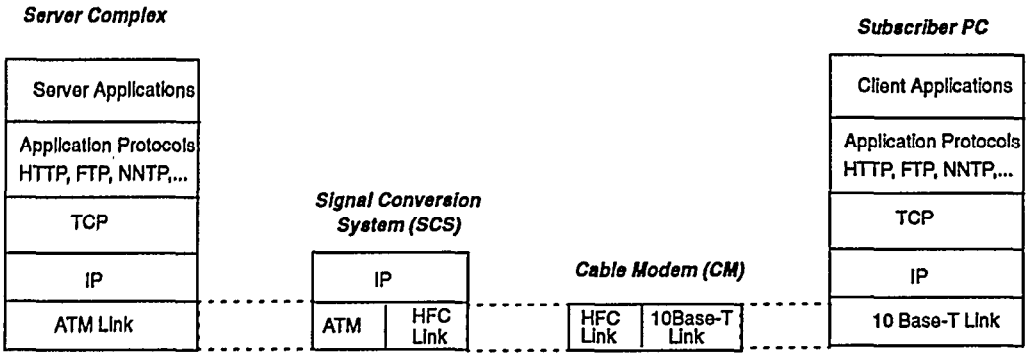



Fig. 2. Protocol layering in the broad-band data system.

HFC network to permit the data channels to operate at different frequencies than analog television channels. The design of the HFC network forces distinct downstream and upstream channels to be used for communication to and from the home, respectively. In most deployments, the downstream channels operate in the 450–750-MHz frequency band whereas the upstream channels operate in the 5–40-MHz band.

In a subscriber's home, access to broad-band data services is enabled through a cable modem (CM) that connects over a 10Base-T interface to a home PC. The CM contains modulation–demodulation hardware to receive and transmit signals over the HFC network. In keeping with the current trend on the Internet, where a majority of the traffic is Web access from various servers, the traffic in the broad-band data system is expected to be predominantly retrievals from the server complex to subscribers' homes. Many CM implementations themselves are asymmetric, offering up to 30 Mb/s for downstream transmission to subscribers' homes and 0.5–4 Mb/s for upstream transmission from subscribers' homes to the server complex.

Fig. 2 depicts the typical protocol layering in the broad-band data system. The client application component executes on the subscriber's PC and communicates with server component(s) in the server complex. The standard Internet protocol (IP) suite is used for communication between subscriber PC's, the server complex, and the Internet. IP packets are transported over ATM between the server complex and the SCS. For communication over the HFC network, the IP packets are encapsulated into HFC link packets.

## III. TCP Performance in HFC Networks

### A. Problem Definition

One of the primary ways of characterizing performance of the broad-band data system as perceived by subscribers is in terms of throughput observed by applications operating above the TCP layer. Since it is an end-to-end performance metric, throughput is dependent on several factors including the configuration and loading of the servers and of subscribers' PC's. From the networking perspective, the achieved throughput depends not only on the bandwidths available on the downstream and upstream channels but also on the specific TCP implementations used at the servers and PC's (e.g., BSD Unix has Tahoe and Reno TCP variants that differ in the congestion control mechanisms they implement and, hence, offer differing throughput), and on the settings of different TCP parameters including the socket buffer size, the maximum segment size used, etc. Two other dynamically changing factors that significantly affect the achieved throughput are the degree of loading of the network and the physical network's noise characteristics, referred to henceforth as *media errors*. Since both of these contrasting network conditions result in the same symptom, i.e., packet loss, differentiating between the two cases is a challenge. TCP has many built-in congestion control mechanisms to handle network overload effectively. However, when operating in a network that is prone to media errors, TCP reacts in the same way as it does during congestion, i.e., it slows down transmissions even though the network is not overloaded. This results in a sharp decrease in TCP's performance [12].

This problem is especially significant in HFC access networks which are prone to a variety of RF impairments owing to defects in home wiring and cracks that appear in the coaxial cables because of exposure to harsh environmental conditions [7]. Since most of the common noise sources operate in the lower frequencies, the upstream channels are especially vulnerable to RF impairments. Our initial experiences with monitoring data services over HFC networks indicate that the

RF impairments at the physical layer usually result in packet losses ranging from 1% to 10%, especially on the upstream channels. In extreme cases, packet losses have been observed to be as high as 50% [12].

### B. Related Work

*1) TCP Congestion Control Mechanisms:* The various mechanisms that current TCP implementations incorporate to adapt to the network conditions are based on [10]. In the initial *slow-start* phase of a TCP connection, the transmitter starts with a transmission window of one packet[1] and then doubles its window during every round trip (this amounts to a growth of the window by one packet for each acknowledgment (ACK) received), until the maximum permissible window size, determined by the connection's socket buffer setting, is reached. This exponential rate of growth of the window may be constrained when the receiver implements a *delayed acknowledgment strategy*. As per this strategy, the receiver holds back the transmission of an ACK for a received packet until a subsequent packet is received, at which time it transmits a *cumulative ACK* to acknowledge both of the received packets. RFC 1122 indicates that the receiver must transmit at least one ACK for every two maximum-sized data packets received [3]. While using delayed ACK's, the maximum period for which the receiver can wait for a subsequent packet is limited to about 200 ms.

To detect packet losses, the transmitter uses the straightforward method of estimating round-trip times of packets. Based on the transmission times of packets and the arrival times of ACK's, the transmitter maintains an estimate of the maximum round-trip time over a TCP connection. When an ACK for a packet does not arrive within the maximum round-trip time (and no ACK's for subsequent packets are received), the transmitter detects the packet loss, retransmits the lost packet, shrinks its transmission window to one, and reverts back to the slow-start phase. The time that the transmitter spends waiting for an ACK of a lost packet to arrive is referred to as a *timeout*. Following a timeout, slow-start happens until the transmission window reaches half the value it had when the timeout occurred. From this point onwards, the transmitter enters the *congestion avoidance* mode, growing its window by one packet for every round-trip time. In this mode, window growth is linear, rather than exponential as in the slow-start phase.

While the above strategy is useful when several packets are lost in a transmission window, the TCP transmitter has a more efficient way to detect and recover from isolated packet losses. Isolated packet losses result in out-of-sequence packet arrivals at the receiver and trigger the transmission of *duplicate ACK's* (dupACK's). Upon receiving three dupACK's, assuming that the packet referred to by the dupACK's has been lost, the transmitter performs fast retransmit by immediately retransmitting the lost data packet. When the ACK for the retransmitted packet is received, the transmitter performs *fast recovery* by shrinking its transmission window to half the value of the window at the time when the packet loss was detected. Then, the transmitter begins to operate in the congestion avoidance mode.

*2) TCP Over Lossy Networks:* The problems in TCP performance over lossy networks have thus far been addressed mainly in the context of wireless local area networks (LAN's). Three main approaches have been proposed for such networks.

- *Reliable link layer protocols:* In this approach, the link layer protocol incorporates retransmission mechanisms to recover from media errors, thereby masking the effect of media errors from the TCP layer above. Most SCS-CM protocols being currently used for communication over the HFC network do not guarantee reliable delivery. Incorporating reliability into these protocols would necessitate changes in the SCS and CM designs. Furthermore, since not all applications require reliability, a common reliable link protocol cannot be used for all applications.
- *"TCP-aware" link layer protocols:* A typical example in this category is the Snoop protocol [2]. As per this approach, a base station in a wireless LAN tracks all TCP connections and maintains a cache of recently transmitted packets. When the base station notices dupACK's, it retransmits packets locally on the wireless segment without the original TCP transmitter (that is on a wired network) even being aware of the loss. This approach is not suitable for large network deployments, in which a wireless base station or an SCS in an HFC network must snoop on all packets that they route and maintain state information for each TCP connection.
- *"Split connection" protocols:* In this approach, a TCP connection between the source on a wired network and destination on a wireless network is transparently split into two transport connections: one for the wired network and another for the wireless network [1]. The TCP implementation for the wireless network is modified so as to be aware of handoffs in the wireless network and to initiate slow-start immediately after a handoff. Although initially designed to handle mobility issues, this approach is also useful in handling packet losses that occur in the wireless network locally [2]. In an HFC network, the split connection approach must be implemented at the SCS, thereby requiring per-connection state information and processing at the SCS.

Many approaches that have been proposed for increasing the performance of TCP during congestion are applicable, to some extent, in lossy HFC networks. The implementation of selective acknowledgment in TCP to enable the transmitters to more precisely determine packets that are lost and recover from such losses is proposed in [11]. Initial testing of the selective acknowledgment feature promises significant performance gains in lossy networks. However, to be useful, selective acknowledgment requires changes in TCP implementations not only in the servers but also in the several hundred thousand subscriber PC's.

The problems with the exponential window increase during the TCP's slow-start phase are highlighted in [9]. In cases

[1] For simplicity, in this paper, TCP's transmission window sizes are expressed in terms of maximum-sized data packets rather than in terms of bytes. Furthermore, packets are assumed to be numbered with consecutive sequence numbers, rather than by the number of bytes contained in each packet.

when the TCP socket buffer setting is very large, the exponential window increase can overwhelm the network routers and lead to timeouts. To overcome this problem, a method for enabling the TCP transmitter to estimate and adapt to the available bandwidth on the network is proposed in [9].

An entirely new variant of TCP, called TCP Vegas, that implements new slow-start and congestion avoidance techniques to adaptively adjust the TCP window upon sensing network congestion is proposed in [4]. Unlike earlier implementations, TCP Vegas uses a fine-grained timer to time every packet and ACK and to accurately estimate the round-trip time. Based on this estimate, TCP Vegas determines, much earlier than TCP Reno or Tahoe, if and when packets should be retransmitted. This proposal requires significant changes to existing TCP implementations and is yet to be adopted in commercial products.

### C. Contributions of This Work

This work analyzes the performance of TCP applications in the unique asymmetric and heterogeneous environment that HFC networks offer. Since different upstream and downstream channels with different noise characteristics are used in HFC networks, the paper studies the relative effect of packet loss and ACK loss on TCP applications and the variations of these effects with data transfer size. Simulations indicate that TCP applications are much more sensitive to loss of data packets than to loss of ACK's and that larger data transfers are likely to be affected much earlier and to a greater extent than smaller transfers.

Focusing mainly on downstream data transfers from the server complex to subscribers' homes, we explore various methods of tuning TCP parameters of existing TCP implementations to ensure better network performance. Since it determines the maximum transmission window that a TCP connection can use, the socket buffer setting directly governs the achieved throughput. Using simulations, we illustrate that proper sizing of TCP socket buffers by taking into account that the buffering capacity of the CM's is critical for high performance in HFC networks. Toward this end, we derive an analytical model for determining the TCP socket buffer size setting. Since the effective socket buffer size is the minimum of the buffer sizes set at the two ends of a connection, the buffer size setting thus determined can be enforced from the local servers without requiring configuration changes to subscribers' PC's.

We also study the impact that delayed ACK implementation in subscriber PC's has on throughput, especially during times when the network is prone to losses, and devise ways to overcome these problems using minor alterations to TCP parameter settings at the local servers. To further increase network performance under losses, we propose ways of tuning the TCP retransmission timeouts and fast retransmit implementations at the local servers to increase TCP's reactivity to loss. These modifications are simple to implement yet highly effective. Moreover, these modifications require only minor changes to TCP implementations at the local network servers alone (and not at subscribers' PC's). Although designed in the context of HFC networks, the modifications are general enough to be applicable to other access technologies, especially wireless, that are prone to media errors.

## IV. Characterizing TCP Performance in HFC Networks

### A. Network Model

In order to characterize the performance of TCP applications in HFC networks, we have developed a model of a typical HFC network using the *ns* network simulator from Lawrence Berkeley Laboratories. The downstream and upstream bandwidths on the HFC network are assumed to be 25 and 3 Mb/s, respectively. Based on experimentations in typical HFC networks, the round-trip delay between the servers and a subscriber PC is set to 20 ms. Since we are concerned mainly with the performance of TCP under losses introduced by media errors, the following simplifying assumptions are made.

- Although an SCS can support multiple downstream and multiple upstream channels, only one downstream channel and its corresponding upstream channel is modeled.
- Since we are interested in TCP performance under media loss rather than under congestion, the precise contention resolution and network access algorithms of the HFC upstream link protocol are not modeled. Furthermore, in the simulations, the number of simultaneous connections is controlled so as to avoid network overload.
- In the absence of precise models for media errors that happen over HFC networks, we model loss of TCP packs and acknowledgment using Poisson distributions. Consequently, the effect of burst losses is captured at high loss rate.

The network traffic is assumed to be predominantly Web and FTP access from the local server complex. All receivers and transmitters are assumed to implement TCP Reno. In keeping with most commercial TCP implementations, the receivers are assumed to implement delayed ACK's. The TCP data packets and ACK's are assumed to be 1460 and 40 bytes in length, respectively. Since different downstream and upstream channels are used over the HFC network, we begin our analysis of TCP performance by considering cases when only TCP ACK's are lost and when only data packets are lost.

### B. Effect of Acknowledgment Loss

Fig. 3 depicts the degradation in TCP performance with loss of ACK's for a 3-Mb data transfer. To study the effect of ACK loss in isolation, the downstream channel is assumed to be lossless. Because TCP uses a cumulative acknowledgment strategy, where each acknowledgment indicates the sequence number of the packet that the receiver expects next, loss of an ACK can be compensated for by the arrival of a subsequent ACK. For instance, when a transmitter receives the third delayed ACK out of a sequence of ACK's numbered 2, 4, 8,$\cdots$, the transmitter can detect loss of the ACK numbered 6. However, since it receives ACK 8 corresponding to data packets 6 and 7, and not a dupACK for packet 6, the transmitter can infer that data packets 4 and 5 (corresponding to ACK 6) were successfully received and therefore need not be retransmitted. From this example, it is clear that loss of

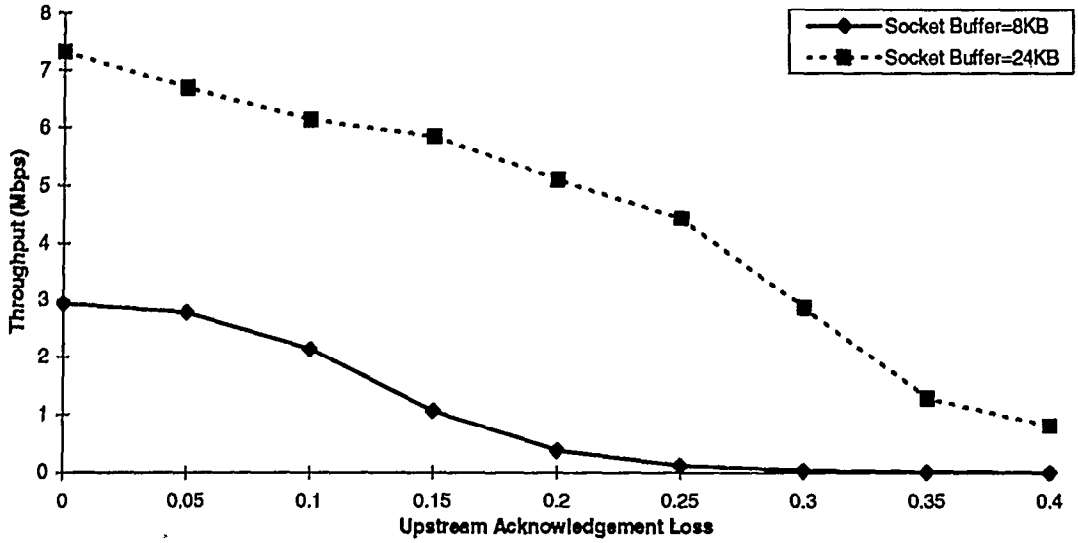



Fig. 3. Effect of ACK loss on performance of a TCP connection transferring 3 Mb of data downstream from a local server.

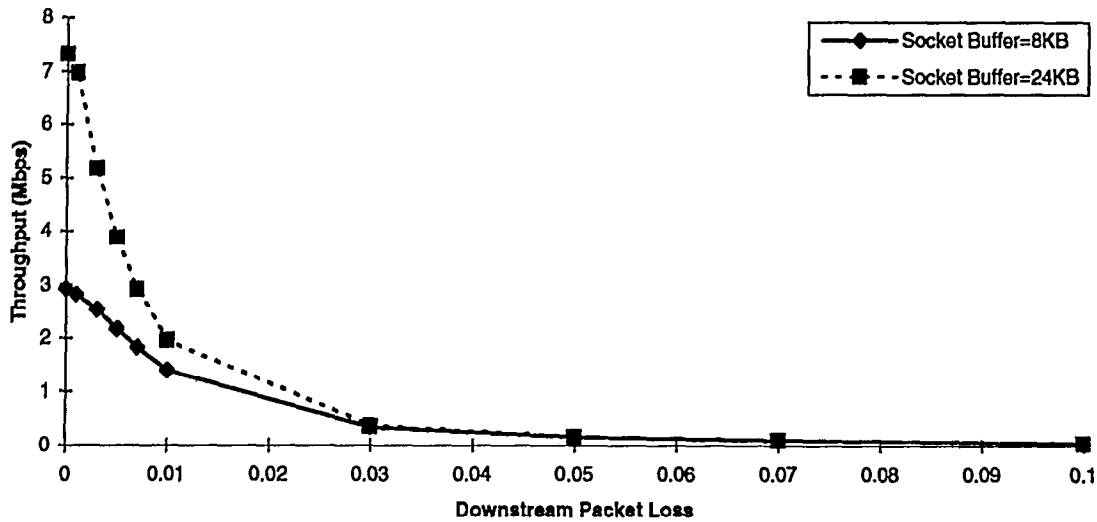



Fig. 4. Effect of downstream packet loss on performance of a TCP connection transferring 3 Mb of data downstream from a local server.

an ACK does not necessarily result in retransmission of data packets. However, an increase in loss of ACK's can have two consequences that reduce throughput.

- *Slowing the transmitter:* When an ACK is lost, the transmitter has to wait for a subsequent ACK to recover from the loss. Frequent ACK loss can introduce a significant waiting time at the transmitter, thereby slowing the transmitter and lowering the throughput. Moreover, since TCP's slow-start and congestion avoidance algorithms increase the transmission window based on ACK's received at the transmitter, loss of ACK's also slows down the window increase, thereby reducing throughput [13].
- *Timeouts resulting in significant throughput reduction:* When the ACK's for all outstanding packets in a transmission window are lost, the transmitter detects the loss only after a timeout. Since most commercial TCP implementations use a coarse granularity (500 ms) retransmission timer to estimate round-trip time, the waiting time before a TCP transmitter retransmits a lost packet, computed as the sum of the mean and four times the maximum deviation in round-trip times, is at least 2–3 s [13]. Since typical data transfer times range from few tens of milliseconds to several seconds, even a single timeout during the lifetime of a TCP connection results in a significant degradation in performance.

Fig. 3 contrasts the performance obtained for a larger TCP socket buffer size. The larger TCP socket buffer size results in a larger transmission window, which in turn results in a greater number of ACK's per transmission window (the total number of ACK's generated during a data transfer remains unchanged). The increase in ACK's in a transmission window reduces the probability of a timeout due to a loss of all of the ACK's associated with the window, thereby increasing the robustness of the TCP connection to ACK loss.

### C. Effect of Data Packet Loss

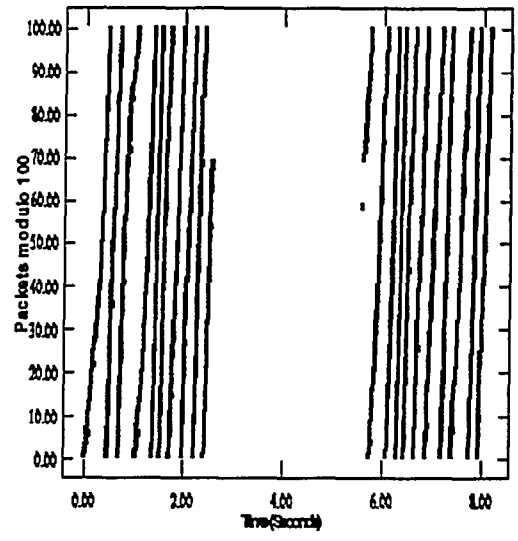
In the previous section we have seen that by using larger buffers for TCP connections, the effect of upstream ACK loss can be mitigated. However, the effect of downstream data packet loss is much too severe to be easily mitigated. Fig. 4

depicts the dramatic reduction in throughput that results when packet loss occurs during a 3-Mb data transfer. To concentrate upon the effect of packet loss, the upstream channel is assumed to be lossless. As is evident from the figure, even a 1% packet loss results in over 50% degradation in throughput for an 8-kb socket buffer. The degradation is even larger for larger socket buffer sizes.

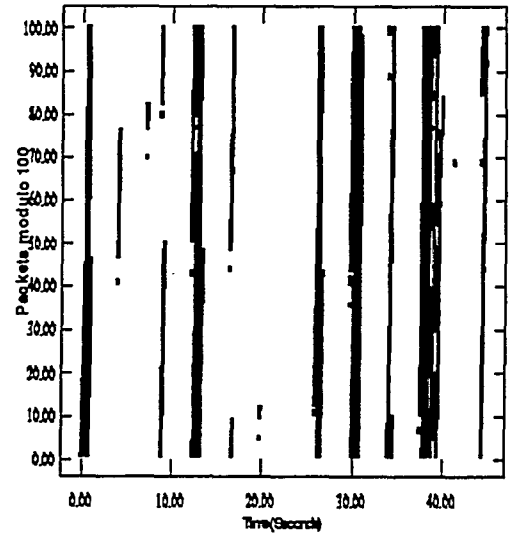
There are several reasons for the dramatic degradation in throughput when data packets are lost. Firstly, unlike in the case of ACK loss, each data packet loss results in one or more data packet retransmissions. When a packet loss occurs, even though TCP may recover from the loss using fast retransmit followed by fast recovery, the TCP transmitter shrinks its current transmission window by half and begins to operate well below the maximum permissible window for several round-trip times. Subsequent periodic losses before the TCP connection reaches its maximum permissible window size can cause the average transmission window to reduce even further. Since TCP's fast retransmit mechanism retransmits a packet only after the sender receives three dupACK's, fast retransmit becomes ineffective when the TCP window falls below 4. At this time, TCP can recover from a packet loss only by means of a retransmission following a timeout. Timeout also happens when multiple packet losses occur in the same transmission window. In such cases, although the transmitter notices the first packet loss and recovers from it using fast retransmit, often following the recovery, TCP's transmission window is not large enough for three dupACK's to be received to recover from a second packet loss [9].

The two consequences of packet loss mentioned above significantly impact throughput. Since they last for several seconds, the impact of TCP timeouts on performance is more adverse than that of fast retransmits. The dramatic reduction in throughput observed in Fig. 4 as packet loss increases is attributable to the significant increase in probability of timeouts with increase in packet loss. To illustrate the impact of timeouts on throughput, Fig. 5(a) and (b) presents simulation traces of a 3-Mb data transfer during times when packet loss is 1% and 3%, respectively. A gap in transmission represents a timeout at the transmitter. Notice that whereas a 1% loss causes just one timeout, a 3% loss causes 11 timeouts, resulting in a fivefold increase in the transfer time.

As is evident from Fig. 4, a larger TCP socket buffer does not improve performance when packet loss is high. This is because frequent packet loss forces TCP to operate with a smaller window and, hence, the connection does not make use of the larger available buffer.




(a)




(b)

Fig. 5. Simulation traces indicating the transmission times of packets during a 3-Mb data transfer for downstream packet loss rates of 1% and 3%. In the latter case, throughput is five times lower than in the former.

### D. Effect of Packet Loss for Different Data Transfer Sizes

Fig. 6 compares the sensitivity of different data transfer sizes to packet loss (their sensitivity to ACK loss follows a similar pattern). To evaluate the effect of packet loss without being biased by the implementation of delayed ACK at the PC's, a modified TCP Reno transmitter is used in the simulation. (The peculiar problems introduced by delayed ACK and modifications to deal with this problem are discussed later, in Section V.) Under no loss conditions, the smaller transfers yield much lower throughput than the larger transfers. This is because TCP starts with an initial window of one packet and requires several round-trip times to grow its transmission window. Once the transmission window reaches its maximum value, throughput achieved remains almost constant independent of the data transfer size.

As Fig. 6 depicts, although packet loss affects data transfers of all sizes, the extent of its impact varies depending on the data transfer size. For the simulated network, throughput degrades much more rapidly for data transfers of 300 kb and above than for the smaller transfer sizes. For instance, when the packet loss rate increases to 5%, the average throughput of 300-kb data transfers drops from 2.8 to 0.2 Mb/s, whereas for 30-kb transfers, throughput only changes from 1.5 to 0.9 Mb/s. To see why this is the case, consider Fig. 7, which contrasts the distribution of throughput values observed during 30- and 300-kb data transfers. The 25 percentile, the median,

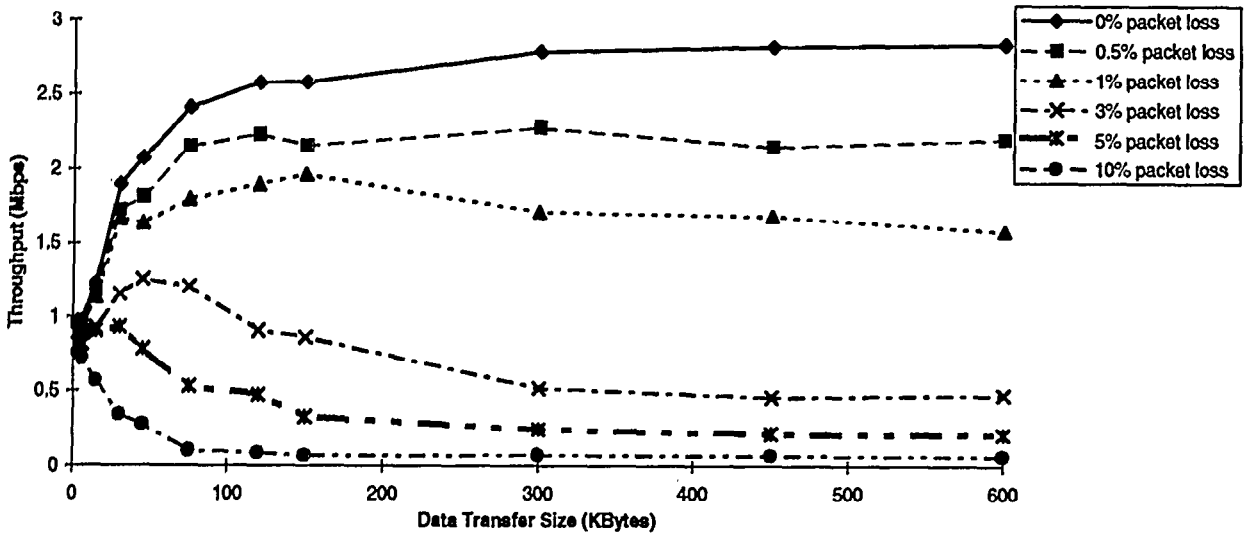



Fig. 6. Effect of packet loss on throughput for different data transfer sizes from a PC with a TCP socket buffer of 8 kb.

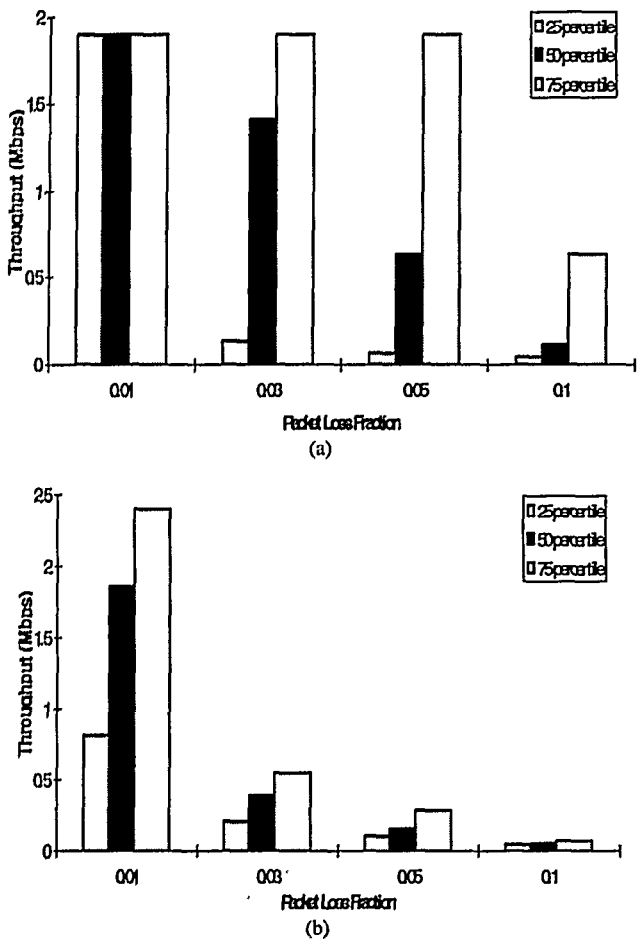



Fig. 7. Distribution of throughput achieved during 100 runs of the simulator for 30- and 300-kb data transfers.

and the 75 percentile values, computed based on 100 runs of the simulator for each loss rate and data transfer size, are shown. The following observations can be made from Fig. 7.

- At a loss rate of 1%, a majority of the 30-kb data transfers are unaffected [Fig. 7(a)]. In contrast, over half of the 300-kb data transfers experience a significant reduction in throughput, and one-fourth of the transfers achieve only a third of the normal throughput [Fig. 7(b)]. This contrasting behavior is attributable to the relative durations of the two data transfers. Since a 30-kb transfer involves transmission of a few packets only (approximately 20 packets of 1.5 kb each), the probability that such a transfer experiences a packet loss is less than the probability that a longer 300-kb transfer involving hundreds of packet transmissions experiences a loss. For instance, when the packet loss rate is 1%, on an average, only one out of five data transfers of 30 kb (20 packets) experiences a loss. Furthermore, because of their smaller duration, the 30-kb transfers are unlikely to experience timeouts because of multiple packet losses. On the other hand, almost every data transfer of 300 kb experiences packet loss. More importantly, the probability of occurrence of multiple packet losses and of timeouts is also correspondingly higher for 300-kb transfers.
- As the packet loss rate increases, 30-kb data transfers too begin to experience the effects of packet loss: a 3% loss reduces the median by 25% and a 5% loss by 66%. However, because of the shorter duration of the transfers, the median and the 75 percentile values for 30-kb transfers are much higher than the corresponding values for 300-kb data transfers. The different distribution of throughput values in the two cases accounts for the higher average throughput of the 30-kb transfers seen in Fig. 6.
- At loss rates of 3% and higher, the 25 percentile throughput values for the 30- and 300-kb transfers are comparable, implying that when packet loss impacts a data transfer instance, its effect on throughput is drastic, independent of the data transfer size.
- Notice also the much larger difference between the 25 percentile and the 75 percentile values for the 30-kb transfers, as compared to the 300-kb transfers. This implies that when packet loss is 3% or more, subscribers transferring 30 kb of data are likely to observe significant variations in throughput. In contrast, at this loss rate almost all 300-kb transfers are likely to feel the impact of packet losses.

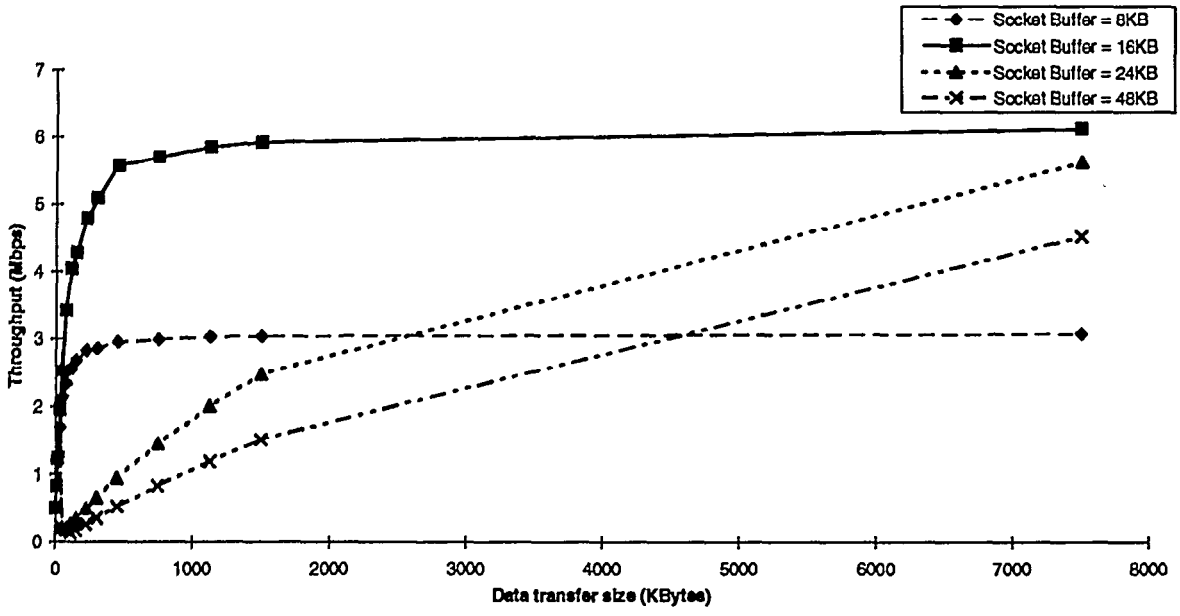



Fig. 8. Effect of CM buffer size on throughput of a TCP connection. In this simulation, there were no media errors at the physical layer. The CM buffer was assumed to be 10 kb.

From a data service operator's perspective, the above discussion indicates that subscribers who invoke larger data transfers are likely to experience the effects of packet loss earlier and to a greater extent than others.

## V. Tuning TCP for Higher Performance by Setting TCP Buffer Size

To enhance TCP performance in lossy HFC networks, in the following sections we consider two complementary approaches. The first approach involves adjusting the TCP socket buffer size to achieve maximum performance. The second approach involves tuning TCP's timeout and fast retransmit mechanisms to reduce the frequency and the duration of timeouts, respectively.

### *A. Effect of Cable Modem Buffer Capacity on TCP Performance*

A larger socket buffer allocation for a TCP connection permits a larger number of packets to be transmitted per unit time, thereby making better utilization of the available bandwidth. Consequently, a larger TCP socket buffer enables a significant increase in throughput when packet loss is not very high (see Fig. 4). As already explained, an increase in the socket buffer size also reduces the TCP connection's vulnerability to ACK loss.

However, in an HFC network, a limit on the TCP socket buffer size is imposed by the buffering available at the CM's. Buffers are provided in a CM to counteract the difference in transmission speeds between the downstream channel on the HFC network (25 Mb/s in our simulated network) and the 10Base-T connection between the CM and the PC (10 Mb/s). Since the experiments described in Section IV were targeted at understanding the effect of media errors, buffering at the CM was assumed to be unbounded. However, in reality, in order to be cost-competitive, most CM implementations are likely to have limited buffering. Fig. 8 depicts the variation in throughput for different data transfer sizes and socket buffer sizes over an HFC network in which the CM's have 10-kb buffers. Since in this experiment we are interested in exploring the effect of bounded buffering at the CM's, the HFC network is assumed to be lossless. Moreover, in this experiment, each CM supports only one TCP connection.

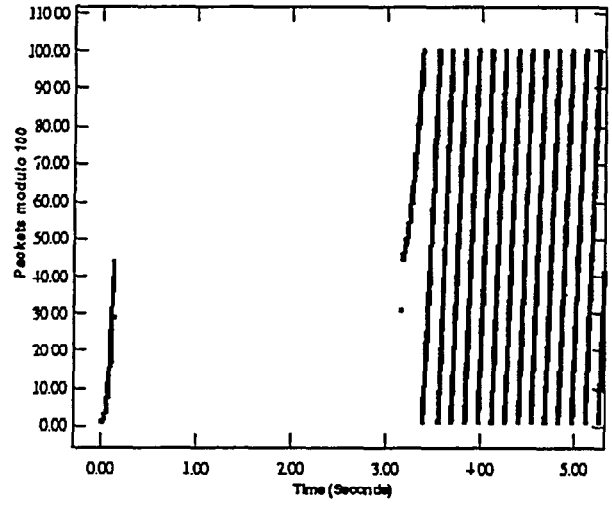



Fig. 9. Illustration of timeout that occurs during the slow-start phase of a TCP connection because the socket buffer size is set to be much larger than the CM buffer size.

As can be seen from Fig. 8, when the socket buffer size increases from 8 to 16 kb, throughput almost doubles for large transfers. However, when the buffer size is further increased to 24 kb, throughput reduces significantly for transfers less than 8 Mb. This drop in throughput is attributable to packet losses that result from buffer overruns at the CM. Recall that during the startup of a connection, in the slow-start phase, the TCP transmitter grows its window exponentially, doubling its window for each round trip. The window increase continues


Fig. 10. Variation in throughput for different TCP socket buffer sizes when three connections simultaneously access the HFC network via a CM with a 10-kb buffer.

either until the window reaches the socket buffer size, or until packet loss occurs. Since it is possible that an entire window of packets could be transmitted consecutively by a local server to the PC, all of these packets could arrive in succession at the CM, at the speed of the HFC downstream channel. Because of the slower speed of the outgoing 10Base-T connection from the CM, packets arriving in succession need to be queued by the CM. When the TCP socket buffer size is much higher than the CM buffer capacity, the CM may not be able to accommodate the arriving packets. Hence, buffer overruns may occur at the CM, resulting in several packet drops. Since the fast retransmit strategy of TCP is not effective when multiple packet drops occur in a window, the transmitter has to recover only after a timeout (see Fig. 9).

Following the timeout, when slow-start is invoked again, the transmitter maintains a threshold that it sets to half the window size at which timeout occurred. When the window size increases to reach this threshold, the TCP transmitter moves into the congestion avoidance mode in which it begins to increase the window linearly, by one packet every round trip. As the window increases, eventually buffer overruns still occur, but owing to the linear increase in the window, only a single packet is dropped at the CM. Consequently, the transmitter recovers from the loss using fast retransmit, instead of having to timeout (see Fig. 9). However, the long initial timeout period significantly impacts the throughput achieved for data transfers of 8 Mb and less for socket buffer sizes of 24 and 48 kb (see Fig. 8). Larger transfers are able to benefit from the larger socket buffer size.

The above experiment highlights the need to consider the CM buffer capacity before setting the socket buffer size for TCP connections. Another factor that must be considered in deciding the socket buffer size is the number of connections simultaneously supported via a CM. Many common Web browsers frequently open multiple simultaneous connections in an attempt to retrieve Web objects in parallel. Fig. 10 illustrates the performance observed when three simultaneous TCP connections, each transferring 300 kb, share the 10-kb buffer of a CM. In this case, packets could arrive over all three connections simultaneously and can result in buffer overruns. For the same socket buffer size, the larger the number of connections, the greater the probability of buffer overruns. In Fig. 10, the smaller socket buffer size performs the best. As the socket buffer size per connection increases, throughput degrades.

### *B. Determining the TCP Buffer Size*

To compute the socket buffer size of a TCP connection, let $B$ represent the TCP socket buffer size and $C$ the buffer capacity of the CM, both represented in terms of TCP packets. Let $P$ represent the maximum TCP packet size in bits. Suppose that only one connection is established via the CM. As explained earlier, the socket buffer size setting that ensures data transfers without timeout must be determined by considering the case when an entire buffer full of data packets are transmitted at the maximum rate to a CM. Ignoring TCP/IP and link protocol header overheads for simplicity, for the simulated HFC network configuration with a downstream channel rate of 25 Mb/s, the time to receive a window full of packets at the CM is $(B \cdot P)/25$ $\mu$s. The CM begins transmission over the 10Base-T connection to the PC connected to it only after the first packet has been fully received from the HFC downstream channel, at which time the available buffering at the CM is $C - 1$. From this time, $B - 1$ packets are received by the CM at the rate of 25 Mb/s and transferred from the CM at the rate of 10 Mb/s. Thus, net accumulation at the CM during this period is

$$\frac{(B-1)\cdot P}{25}\cdot(25-10)\ \text{Mb}.$$

To avoid any buffer overruns at the CM, the necessary condition is

$$C \geq \frac{(B-1)\cdot P\cdot 15}{25}+1.$$

Since timeouts occur during slow-start only when multiple packet drops occur in a window, the above condition can be relaxed by permitting at most one packet drop in a window. This yields the relation

$$C \geq \frac{(B-1)\cdot P\cdot 15}{25}.$$

In the simulated network, for a CM buffer of 10 kb (seven packets), the maximum socket buffer size can be computed from the above equation to be 19 kb, which matches the results from Fig. 8.

When there are $n$ connections supported by the same CM, in order to guarantee that timeout does not occur, in the pessimistic case, no more than one packet loss should occur during buffer overflow. This leads to the condition

$$C \geq \frac{(B \cdot n - 1) \cdot P \cdot 15}{25}.$$

From the above equations, the TCP socket buffer size setting can be computed based on the buffering at the CM and the number of simultaneous connections to be used. Since the per-connection throughput is directly related to the TCP socket buffer size setting, the average rather than the maximum number of simultaneous connections can be used in the computation above. Alternatively, to ensure a minimum throughput per connection, a data service operator may wish to impose a restriction on the maximum number of simultaneous connections supported from each subscriber PC. Since the effective socket buffer size used for a TCP connection is the minimum of the values supported at each end of the connection, the socket buffer size computed above can be enforced by setting the buffers of connections initiated from the local servers to the above value, without the need to modify subscribers' PC configurations.

## VI. Tuning TCP for Higher Performance by Reducing the Effect of Timeouts

Having determined the optimal buffer size setting for a TCP connection, we now explore various facets of TCP that can be modified in order to reduce the effect of TCP timeouts.

### A. Effect of Delayed ACK

As indicated in Section IV, one of the causes of TCP timeouts is loss of several successive ACK's. Although increasing the socket buffer size can reduce the probability of timeouts due to ACK losses, as seen in Section V, the CM buffer capacity imposes a limit on the TCP socket buffer setting. In this section, we explore an approach for minimizing the effect of ACK losses. Toward this end, we study the effect that support for delayed ACK in TCP implementations of subscriber PC's has on performance.

Even in the absence of packet loss, we have observed that delayed ACK implementation in the TCP receivers has an adverse effect on performance of Web and FTP applications in HFC networks (and in conventional LAN's as well). In most TCP implementations, following the establishment of the TCP connection, the transmission window of the host initiating the connection is one packet and that of the other end of the connection is two packets. This is because although both ends of the connection start with an initial window of one packet, the last ACK in the TCP three-way handshake sequence (SYN/SYN_ACK/ACK) causes the window of the host accepting the TCP connection to increase by one packet.

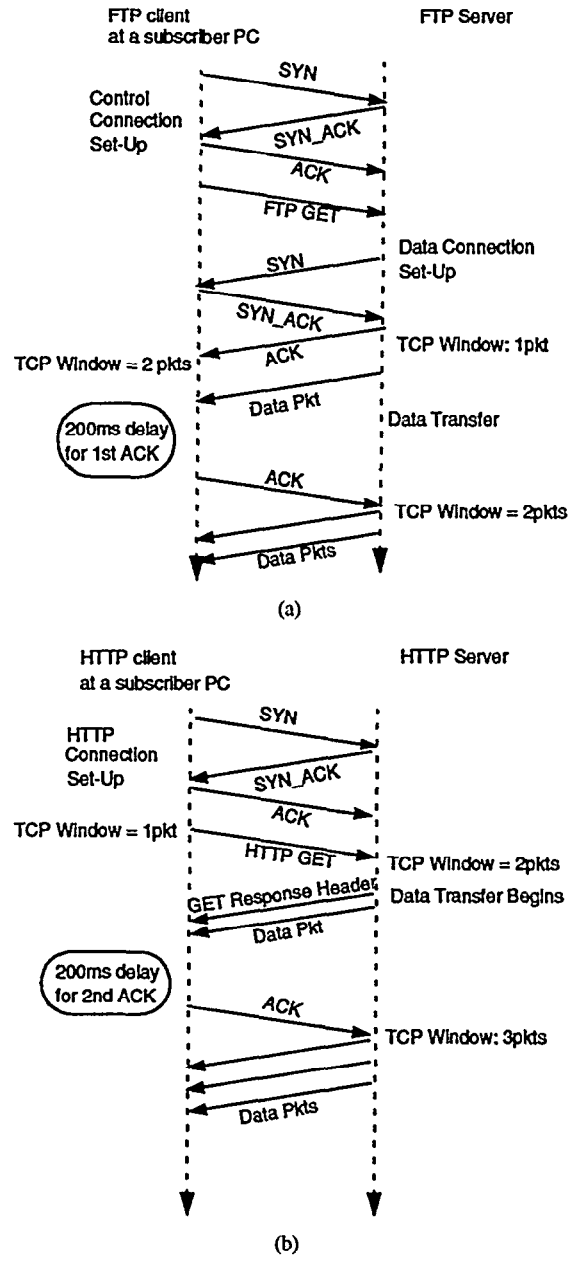



Fig. 11. Illustration of the effect of delayed ACK implementation in TCP stacks of subscriber PC's on performance of (a) FTP and (b) HTTP data transfers. In both cases, the initial delay of up to 200 ms for the first ACK from the subscriber PC reduces throughput by over 50% for data transfers sizes of up to 150 kb.

The difference in the initial TCP windows has different implications for FTP and HTTP applications.

- As illustrated in Fig. 11(a), when a subscriber starts an FTP session with a server, a control connection is first established between the subscriber's PC and the server. *Get* and *put* requests for files are transmitted from the subscriber PC to the server over the control connection. The server then sets up separate TCP connections for each *get* or *put* request. Since the server is the connection initiator, its initial transmission window is one packet. When data is transferred from the server to fulfill a *get* request, the server starts off by transmitting one packet and waiting for an ACK. Since the TCP stack in

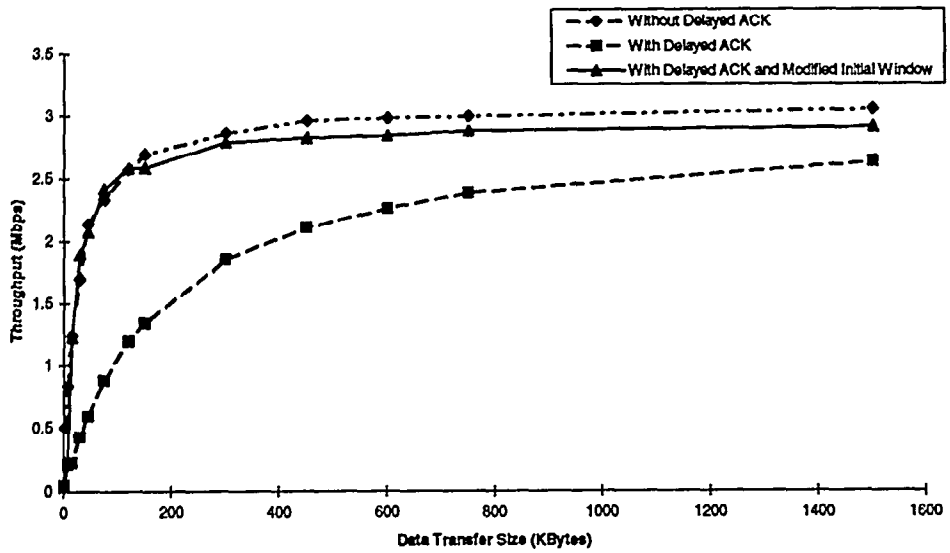



Fig. 12. Illustration of the deleterious impact that delayed ACK implementation at subscriber PC's can have on throughput of HTTP and FTP data transfers. Using an initial window of two packets for the TCP slow-start implementation at the local servces avoids the throughput degradation for data transfers of 150 kb and less.

the subscriber PC supports delayed ACK, the subscriber PC waits for up to 200 ms for subsequent packets before generating an ACK for received packet. This long initial wait time significantly impacts performance. Fig. 12 illustrates that delayed ACK implementation at subscriber PC's reduces throughput by more than 50% for transfers below 150 kb and by less than 10% for transfers of 5 Mb and more. Note that because the initial transmission window of the subscriber PC is two packets, *put* requests do not experience this problem.

- HTTP requests experience a different problem. In this case, the subscriber PC first initiates the TCP connection to the server and transmits the HTTP request either to get or post data. Unlike FTP, data transmission to or from the server using HTTP occurs over the same connection. Since the subscriber PC initiates the TCP connection, the server has an initial window of two packets. Get requests, which are the most common form of HTTP access, may not experience the initial 200-ms delay if the server fully utilizes its transmission window. This is because the server can transmit two maximum-sized packets, thereby forcing an immediate ACK from the subscriber PC. However, many HTTP server implementations (e.g., NCSA 1.5, Netscape Enterprise Server) transmit the HTTP response header separate from the data. Typically, the header is the first packet transmitted by the HTTP server in response to a request and is much smaller than the maximum-sized TCP data packet. Because of the transmission window restriction and the size of the data read from the disk (typically 8 kb), following the HTTP get response header, the server can transmit at most one other maximum sized data packet before it has to wait for an ACK from the subscriber PC. Since it receives less than two maximum sized packets worth of data, the subscriber PC delays its ACK, thereby resulting in an initial delay of upto 200 ms for all HTTP *get* responses.

  A similar problem is described in [8] in the context of persistent HTTP connections. [8] proposes an application modification to solve this problem, by using buffered I/O to ensure that the HTTP response header is never sent out separately. However, this approach not only requires changes to all application servers, but may not also be incompatible with existing Web browsers that expect to receive the header separately. Furthermore, this solution does not resolve the performance problem for FTP applications.

A more general way to avoid the above problems without requiring any change in the application servers is to increase the initial setting of the TCP window size used during slow-start at the local servers alone to two packets (see Fig. 12).

Another drawback of delayed ACK is the reaction it induces in TCP connections when media errors cause ACK loss. Fig. 13 illustrates that for a 3-Mb data transfer, when upstream ACK loss happens, a receiver that does not implement delayed ACK remarkably outperforms a receiver that implements delayed ACK. The difference in performance is attributable to the 50% reduction in number of ACK's when a receiver implements delayed ACK, since only one ACK is transmitted for every two received packets. This reduction in ACK's makes the TCP connection more vulnerable to experiencing timeouts because of loss of all of the ACK's in a window. Furthermore, even in cases when timeouts do not occur, because of the reduction in number of ACK's, the transmitter has to wait for longer time periods for an ACK that follows a lost ACK. The large degradation in performance observed in Fig. 13 indicates the need to increase the frequency of ACK's during communication over lossy HFC networks.

Avoiding delayed ACK at the receiver does not have the same dramatic effect when the losses occur in the downstream channels. This is because TCP Reno does not delay its ACK's when it notices packets arriving out of sequence. In contrast, since the earlier TCP Tahoe implementation continues to delay ACK's even after it notices a packet loss, PC's that implement

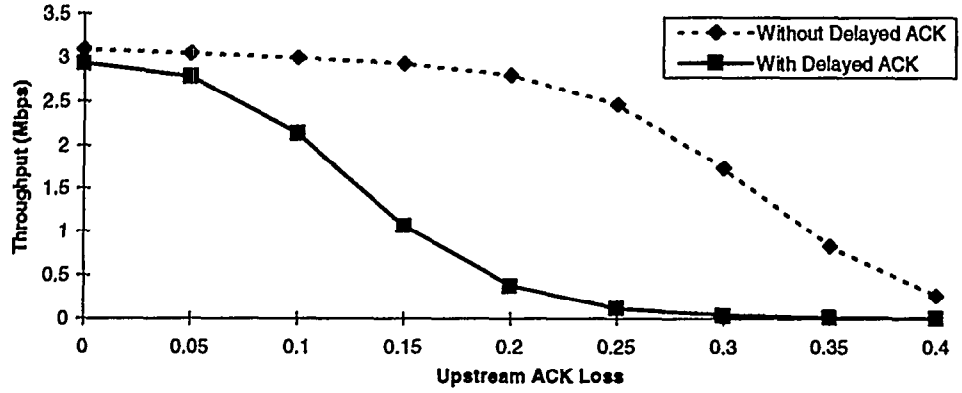



Fig. 13. Implementation of delayed ACK increases a TCP connection's vulnerability to ACK loss. This example involves a 3-Mb data transfer over a connection with maximum window size of 8 kb.

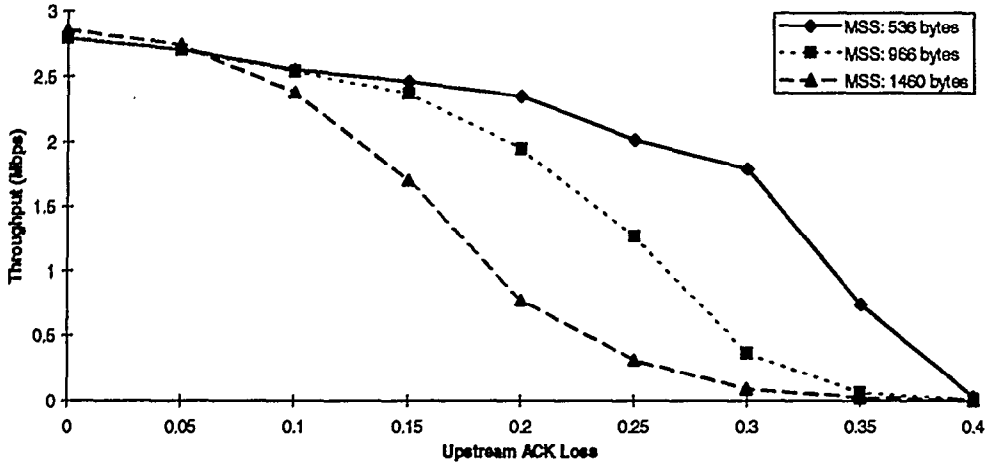



Fig. 14. Throughput variation with ACK loss for different MSS values for a 3-Mb data transfer. By using a smaller MSS, a server can reduce the effect of ACK loss on throughput.

TCP Tahoe are likely to see a significant improvement in performance if delayed ACK is not used.

Avoiding the usage of delayed ACK requires changes in TCP implementations of the several hundreds of thousands of subscribers' PC's, an enormous and unrealistic task. Moreover, applications such as telnet that transmit small TCP packets frequently in both directions over a TCP connection benefit significantly from delayed ACK implementation. In the next section, we discuss changes to TCP implementations at the local servers that can overcome the drawbacks of using delayed ACK in the PC's for bulk transfer applications.

### *B. Setting TCP MSS Values*

Observe that the number of ACK's transmitted from a TCP receiver in a round-trip time is determined not only by the socket buffer size but also by the maximum size of each data packet, referred to as the maximum segment size (MSS). The smaller the MSS value, the larger the number of ACK's in a window. To increase the robustness of TCP connections to ACK loss, it is more efficient to use smaller MSS values during periods of loss. Fig. 14 illustrates the improvement in performance that can be obtained from reducing the MSS even when the TCP receiver implements the delayed ACK strategy. Importantly, since the MSS value for a downstream data transfer is determined by the local server itself, TCP connections originating from the local server can be made more robust to upstream ACK loss by configuring the server to use a smaller MSS.

There are, however, several tradeoffs that must be considered when deciding whether the smaller MSS value should be used always or only during times of significant upstream loss. On the positive side, besides offering greater resilience to ACK loss, the smaller MSS may also decrease the packet loss rate (e.g., depending on the burstiness of errors, if the loss probability of 1-kb packets is 10%, the loss probability of 0.5-kb packets will be between 5% and 10%). Another advantage of smaller MSS is that a larger number of ACK's are generated by the receiver, which increases the probability that the transmitter receives three dupACK's necessary for fast retransmit to be triggered following a packet loss. On the negative side, a smaller MSS results in greater TCP/IP and link protocol header overhead. Furthermore, TCP's slow-start and congestion avoidance mechanisms are slowed down for the smaller MSS since TCP's slow-start and congestion avoidance mechanisms increase the transmission window in terms of packets, irrespective of the MSS in use.

### *C. Using a Finer Granularity Retransmission Timer*

An obvious improvement to enable TCP to react more quickly to network losses is to use a finer granularity timer

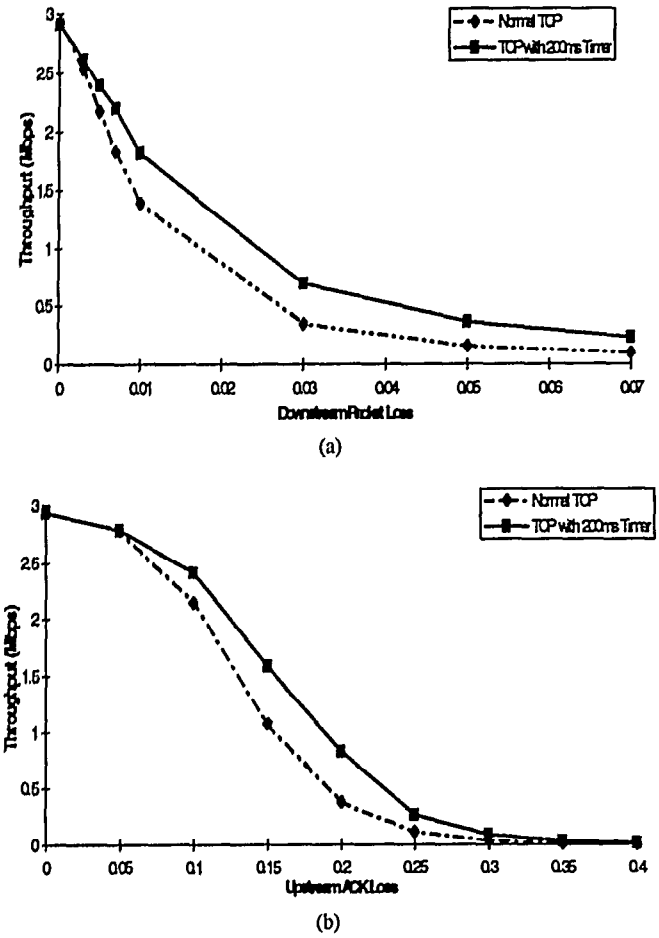



Fig. 15. Increase in performance from increasing the granularity in TCP's retransmission timer to 200 ms (a) during downstream packet loss and (b) during upstream ACK loss. In this example, a 3-Mb data transfer was initiated from a local server to a subscriber's PC.

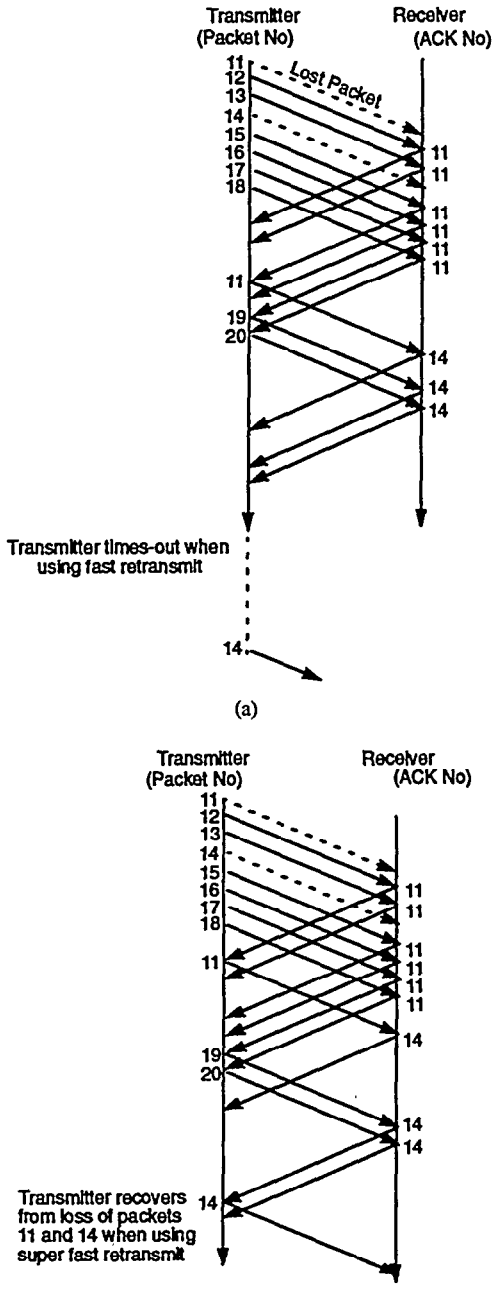



Fig. 16. Demonstration of the advantages of super fast retransmit when multiple packet losses occur in a transmission window of size eight packets. (a) Fast retransmit causes a timeout when packets 11 and 14 are lost. (b) The fast retransmit causes a timeout when packets 11 and 14 are lost. (b) Super fast retransmit is able to recover from the two packets.

than the 500-ms timer used in most current implementations. This would enable the TCP transmitter to obtain a tighter upper bound on the round-trip times, which in turn results in a reduction in the time that elapses before the transmitter times out and retransmits a lost packet. Current TCP implementations use two types of timers: a 200-ms timer that is used for supporting delayed ACK and a 500-ms timer that is used for computing round-trip times. Using the 200 ms timer to estimate round-trip delays as well can increase TCP's reactivity without overly increasing its implementation overhead. As illustrated in Fig. 15(a) and (b), this modification in the timer granularity improves performance both for upstream and downstream losses. The degree of improvement achieved depends on the percentage of losses experienced over the network. For example, when network loss on the downstream is 3%, this simple modification yields an almost twofold increase in performance.

### *D. Using "Super" Fast Retransmit*

A further improvement in performance can be obtained by tuning TCP's fast retransmit mechanism to HFC networks. Since in-sequence delivery of packets is guaranteed over the HFC network and all the way to the local server complex, the receipt of the first dupACK at the server is a clear signal that a data packet has been lost. However, since TCP has been designed to operate in more general networks, where packets may be routed via different paths and may therefore arrive out of sequence, the TCP transmitter waits for three dupACK's to arrive before retransmitting a missing packet. In networks where packets are guaranteed to be received in order, this strategy of TCP unnecessarily delays the detection and retransmission of lost packets. Even more significantly, this strategy makes TCP more vulnerable to timeouts during periods of high loss—since at least three dupACK's are necessary to trigger a fast retransmit, TCP

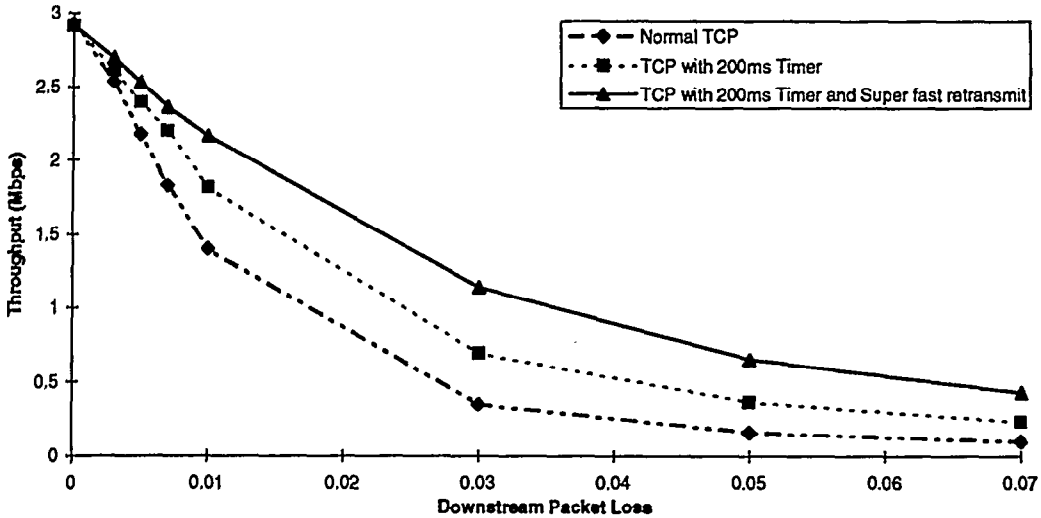



Fig. 17. The complementary effects of increasing the retransmission timer granularity and invoking super fast retransmit for a 3-Mb data transfer between a local server and a subscriber's PC.

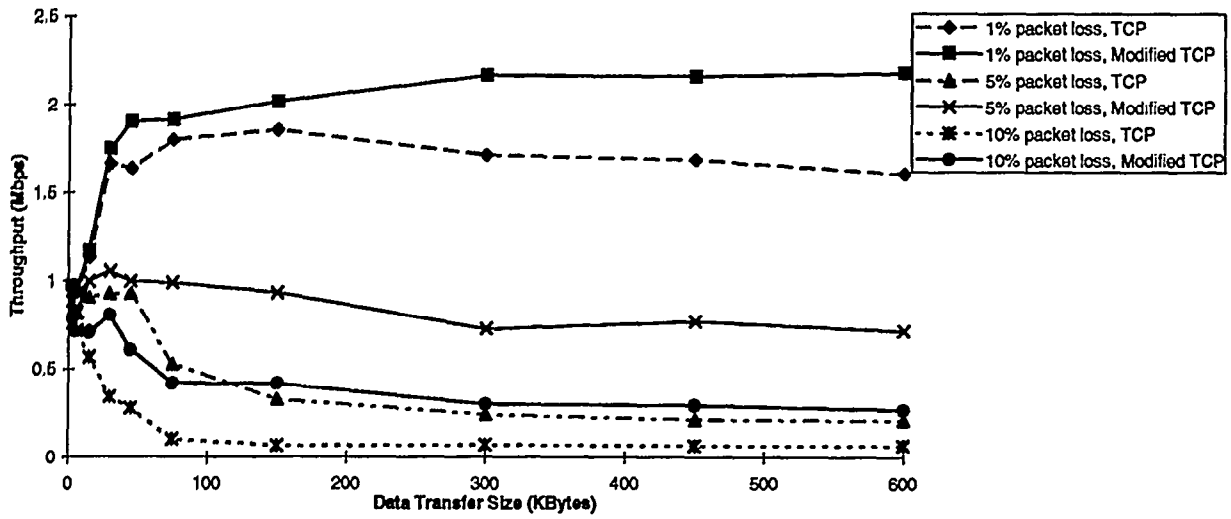



Fig. 18. Illustration of the performance improvements that modified TCP offers for different data transfer sizes and different data packet loss rates.

requires the operating window to be at least four packets for the fast retransmit mechanism to be effective. During periods of significant loss, TCP's operating window drops many times below four, and a single packet loss during the time when the window stays below four (which could be up to two round-trip times) results in a timeout.

We propose to modify TCP implementations at the local servers of HFC networks so that fast retransmit is triggered after the first dupACK is received at the server. By initiating retransmission earlier, this approach, referred to as "*super fast retransmit*," speeds up the recovery of TCP from isolated packet losses. Much more significantly, super fast retransmit reduces the probability of timeouts. Since it requires only a single dupACK to retransmit a packet, super fast retransmit can be triggered even when the window size drops to two and a packet loss occurs. In addition, super fast retransmit increases the possibility that the TCP transmitter detects and recovers from multiple packet losses in a transmission window, as demonstrated in Fig. 16. In this example, when two packets, 11 and 14, are lost, a transmitter that implements fast retransmit experiences a timeout for packet 14 since it receives only two dupACK's for this packet. In comparison, a transmitter using super fast retransmit retransmits packet 14 soon after receiving the first dupACK and is thereby able to recover from the packet losses without experiencing a timeout.

The effects of increasing the retransmission timer granularity and employing super fast retransmit are complementary. Whereas the former reduces the duration of each timeout, the latter reduces the number of timeouts. Fig. 17 illustrates the performance improvement that can be obtained from using a TCP implementation with these modifications during different network loss conditions for a 3-Mb data transfer.

Fig. 18 illustrates the performance improvements for different transfer sizes. For the loss rates shown in the figure, the modified TCP performs uniformly well for data transfers of 50 kb and above, offering performance improvements ranging from 25% to 300%. For smaller transfers of 50 kb and less, the effect of modified TCP is less dramatic for loss rates of 5% and less. At higher loss rates, the performance improvements even for small transfer sizes are very significant. For instance,

for a 30-kb data transfer at a 10% loss rate, modified TCP doubles the achieved throughput.

## VII. Implications for Broad-Band Data Service Operators

In this section, we discuss some implications of our analysis for HFC network operators and equipment manufacturers.

- *Viability of data services even under harsh upstream network conditions:* In recent times, critics have doubted the viability of providing data services over HFC networks, mainly on account of the high error rates on the upstream channels [6]. Our analysis indicates that for downstream access, TCP connections can be tuned to be highly robust to loss of ACK's on the upstream channel, so that acceptable performance can be achieved even under harsh upstream conditions. For example, with the enhancements proposed in the previous sections, even a 20% loss of ACK's on the upstream channels results in less than 10% reduction in TCP performance.
- *Need to optimize performance by careful design and maintenance:* To the many who have believed that data communication protocols are tolerant to loss, our analysis has demonstrated that a low (1%) downstream packet loss can degrade performance by up to 50%. To meet subscriber expectations, data service operators must strive to tune and maintain the physical network to reduce RF impairments. Considering the predominance of downstream data transfers, the downstream channels must be especially carefully tuned. Forward error correction and bit interleaving techniques incorporated in the CM's can mask physical layer errors from the TCP layer. Proactive monitoring can alert operators about impairments before subscribers notice the problem.

  In the future, when applications transferring data upstream become commonly used, upstream error rates must be strictly controlled. Based on our analysis, we conjecture that upstream packet loss, more than the limited upstream spectrum available, is likely to limit the achievable throughput. Channel error sensing and frequency agility capabilities in the CM's can help in sustaining high performance levels on the upstream channels.
- *Implications for CM-SCS architectures:* In many first generation architectures, like the CM's, the SCS too transmits on the upstream frequencies and a separate frequency translator is used to reflect upstream transmissions on the downstream channels. Since noise on the upstream channels is also reflected downstream, in such CM-SCS architectures, even transmissions from the server complex to subscribers' PC's may be affected by upstream noise in the HFC network. Our analysis indicates that CM-SCS architectures which separate upstream transmissions from CM's and downstream transmissions from the server complex are likely to yield much higher performance for TCP traffic.
- *Tuning TCP socket buffer settings for optimal performance:* Until now, very little attention has been placed to tune the TCP connection parameters to the specific CM architecture. We have demonstrated the performance advantages that can be obtained simply by tuning the TCP socket buffer parameters based on the buffering capacity of the CM's and the number of simultaneous connections supported by the CM.

We are beginning to implement and experimentally evaluate the impact of our proposed modifications in real-world HFC networks.

## References


[1] A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. 15th Int. Conf. Distributed Computing Systems (ICDCS)*, May 1995.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proc. 1st ACM Conf. Mobile Computing and Networking*, Berkeley, CA, Nov. 1995.

[3] R. Braden, "Requirements for Internet hosts communication layers," Internet Request for Comments RFC 1122, Oct. 1989.

[4] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1465–1480, Oct. 1995.

[5] J. Dail, M. Dajer, C.-C. Li, P. Magill, C. Siller, K. Sriram, and N. Whitaker, "Adaptive digital access protocol: A MAC protocol for multiservice broadband access networks," *IEEE Commun. Mag.*, vol. 34, pp. 104–112, Mar. 1996.

[6] J. Dvorak, "The looming cable modem fiasco," *PC Mag.*, Sept. 1995.

[7] C. Eldering, N. Himayat, and F. Gardner, "CATV return path characterization for reliable communications," *IEEE Commun. Mag.*, vol. 33, pp. 62–69, Aug. 1995.

[8] J. Heidemann. (October, 1996) "Performance interactions between P-HTTP and TCP implementations." *Computer Commun. Rev.* [Online]. Available: HTTP: http://www.isi.edu/div7/lsam/publications/phttp_tcp_interactions/.

[9] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," in *Proc. ACM SIGCOMM'96*, Palo Alto, CA, Aug. 1996, pp. 270–280.

[10] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM Symp.*, Aug. 16–19, 1988, pp. 314–329.

[11] M. Mathis and J. Mahdavi, "Forward acknowledgment: Refining TCP congestion control," in *Proc. ACM SIGCOMM'96*, Palo Alto, CA, Aug. 1996, pp. 281–292.

[12] E. Perry and S. Ramanathan, "Network management for residential broadband interactive data services," *IEEE Commun. Mag.*, vol. 34, pp. 114–121, Nov. 1996.

[13] W. Stevens, *TCP/IP Illustrated*, vol. 1. Reading, MA: Addison-Wesley, 1994.



**Reuven Cohen** (M'92) received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the Technion, Israel Institute of Technology, Haifa, Israel, in 1986, 1988, and 1991, respectively.

From 1991 to 1993 he was with IBM T. J. Watson Research Center, working on protocols for ATM and high-speed LAN's. Since 1993 he has been with the Department of Computer Science, Technion, Israel Institute of Technology, Haifa, Israel. His most recent work focuses on the design and evaluation of protocols for routing, multicast, and transport.



**Srinivas Ramanathan** received the B.Tech. degree in chemical engineering from Anna University, Madras, India, in 1988, the M.Tech. degree in computer science and engineering from the Indian Institute of Technology, Madras, India, in 1990, and the Ph.D. degree in computer science and engineering from the University of California, San Diego, in 1994.

He is currently a Research Staff Member at Hewlett-Packard Laboratories, Palo Alto, CA, where his research focuses on measurement and management for emerging broad-band networks, Internet technologies, and services.