DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume **52**, 2000

The Design and Performance of Mobile TCP for Wireless Networks

Zygmunt J. Haas and Abhijit Warkhedi

{haas, warkhedi}@ee.cornell.edu

School of Electrical Engineering, Cornell University

Abstract – Wireless environments are characterized by lossy links and intermittent connectivity. In the past, considerable research has been conducted to improve the end-to-end performance of TCP over wireless networks. The focus of our research is on improving the end-to-end performance, while minimizing processing load of transport protocols on power-limited mobile devices. In this paper, we present the design and implementation of a transport protocol, called MTCP, for wireless networks. The primary goals of MTCP are as follows: a) alleviate the effects of wireless losses on end-to-end performance, b) minimize processing overhead on mobile devices, and c) allow seamless integration of mobile applications into existing internetworks. Our approach is based on the split-connection model, where the end-to-end connection between a mobile and a stationary host is partitioned into two connections: the connection between the mobile host and the mobile-gateway (or the basestation), and the connection between the mobile-gateway and the fixed host. The division allows us to develop MTCP, which is an optimized protocol emulating TCP functionality on the wireless link. The protocol is designed to be lightweight, in order to minimize and eliminate the unnecessary overhead placed on mobile devices. Our experiments with MTCP on different testbeds indicate a substantial improvement in throughput over the "regular" TCP under varied unreliable operational conditions of the wireless link. In addition, the results show that MTCP is much more efficient in utilizing CPU resources than TCP.

2000 *Mathematics Subject Classification.* Primary 68M12, 68M10.

1. Introduction

The vision behind mobile computing [1, 2] is to support ubiquitous access to various forms of information such as data, voice and (possibly) video. Integration of mobile devices into the existing internetwork consisting of stationary hosts has played a critical role in bringing the state-of-the-art a step closer to realizing this vision. At the same time, continued use of legacy technology in conjunction with wireless networks has given rise to certain problems due to the special characteristics of the wireless link and the requirements of the portable power-limited mobile devices.

One problem is the inability of existing end-to-end protocols such as TCP [3, 4] to effectively cope with highly unreliable wireless links. TCP is specifically designed to perform well in networks where losses occur mostly due to congestion problems. As a consequence, TCP fundamentally assumes that all packet losses in the network are caused due to congestion-related problems. This assumption, however, is incompatible with the properties of a wireless link where losses are due to handoffs and transmission errors caused by noise, interference, and channel fading.

A TCP sender identifies losses either by the arrival of three duplicate acknowledgments or by the absence of an acknowledgment within a timeout period. Upon identifying a loss, the TCP sender invokes various congestion control and avoidance mechanisms such as reducing the congestion window size and/or backing off the retransmission timer [3, 4]. These measures, however, cause an unnecessary reduction in the overall end-to-end throughput in networks consisting of links where packet losses are not due to congestion [5]. Several schemes have been proposed in an effort to alleviate the effects of non-congestion-related losses on TCP performance in wireless or similar high loss links [6, 7, 8].

Another problem posed by the existing communication protocols is the processing requirements placed on the mobile devices. As portable machines shrink in their physical parameters (weight and size), their computing and power

capabilities are also reduced, especially due to the reduced capacity of the power storage devices. Thus, to maintain a constant level of performance of mobile applications, there is a need to reduce the processing load of communication protocols on mobile devices [9]. The focus of our research is to devise a new transport protocol that improves the end-to-end performance of TCP, while reducing the processing requirements on the mobile host. We base our approach on the split-connection model, where the transport-layer connection between a mobile and corresponding stationary host is partitioned into two connections: the connection between the mobile host and the mobile-gateway (also referred to as basestation), and the connection between the mobile-gateway and the corresponding host. The first connection is called the *wireless* segment, while the second portion is the *wired* segment. The division, by itself, is not a new idea and was already employed by the Indirect TCP (I-TCP) [10]. However, the contribution of this study is the design of a lightweight transport protocol, called MTCP, on the wireless segment that not only improves end-to-end performance, but also reduces the processing load on the mobile device, as well as the communication overhead on the wireless segment. Most of our protocol enhancement stems from the important fact that the connection between the mobile host and the mobile-gateway is a point-to-point link and that all the packets arrive in order. Consequently, as we will see in this paper, many functions can be either simplified or totally eliminated in the wireless segment of the transport-layer connection, leading to a lean protocol code on the mobile machine [9].

Following the split connection model employed by I-TCP, we divide the end-to-end logical connection into the MTCP connection communicating over the wireless segment and the "regular TCP" connection communicating over the wired segment. A transport-layer gateway, called the *Redirector*, is implemented on the mobile-gateway to seamlessly integrate both of these connection segments in such a way that the indirection is transparent to the mobile and the corresponding stationary hosts. This paper will primarily focus on the design and

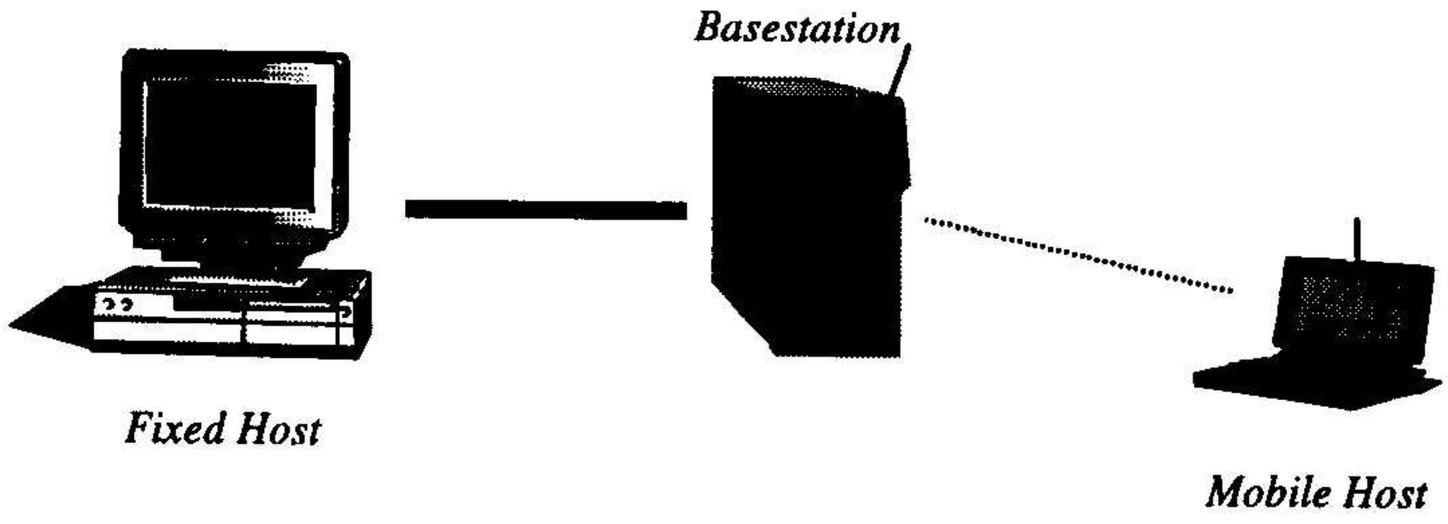

Figure 1: Wireless Network Model

implementation of the MTCP protocol and not on the details of the Redirector, as it has been well studied as part of the research on I-TCP [6].

In addition to designing the MTCP protocol, we present the results of experiments on our testbed under varied unreliable conditions. We evaluate the performance of our protocol using performance metrics such as throughput and CPU-usage. Our measurements, taken at different bit-error rates and raw data rates, indicate a substantial improvement in performance over Indirect TCP and end-to-end TCP.

The rest of this paper is organized as follows. Section 2 briefly describes and compares some previously proposed solutions to the problem of reliable transport protocols over wireless links. Section 3 presents the design of our MTCP protocol. In Section 4, we discuss some details of our protocol implementation. Section 5 presents the results and analysis of several experiments. A discussion on miscellaneous issues is included in section 6. We discuss our future plans in section 7 and conclude with a summary in section 8.

2. Related Work

Figure 1 shows a typical network model for wireless networks. There are fundamentally two different approaches to improving TCP performance in such networks. The first approach attempts to hide non-congestion-related losses from the TCP sender. The intuition behind this approach is that these losses are local to the wireless link and that the problem should be solved locally. The second

approach explicitly notifies the TCP sender of non-congestion-related losses. This allows the TCP sender to differentiate between congestive and non-congestive losses and thereby to react accordingly. In general, we can broadly classify the above two approaches into three categories: a) link-layer scheme, b) split connection scheme, and c) end-to-end scheme. The link-layer and split connection methods belong to the first approach where the problem is solved locally, whereas the end-to-end scheme follows the second approach of explicitly notifying non-congestive losses back to the TCP source [11]. In the following, we provide a more in-depth look at these schemes.

2.1 Link-Layer Schemes

The goal of link-layer schemes is to perform link-layer retransmission on the wireless or similar lossy links. By doing so, most packet losses can be recovered and therefore shielded from the TCP sender. Two main classes of techniques are usually applied by these schemes: automatic retransmission requests (ARQ) and error correction, such as Forward Error Correction (FEC). The main advantage of this approach is that it fits well into the layered structure of network protocols. In addition, the link-layer approach operates independently of higher-layer protocols and does not maintain any per-connection state. The main concern, however, is that most of the link-layer mechanisms compete with reliable transport protocols, such as TCP, in order to recover from losses. This interaction can lead to degraded performance of higher-layer protocols. Furthermore, most link-layer schemes are poor in shielding the TCP sender on the fixed host from wireless losses, since the schemes do not attempt to either deliver retransmitted packets in-order or suppress duplicate acknowledgments sent by TCP on the mobile host [11]. This makes link-layer schemes especially ineffective in dealing with handoffs. The other concern is the additional processing burden that these schemes place on the mobile devices, which are already power and computing limited. Specifically, the ARQ mechanisms may use timeouts to perform retransmissions, while timeouts are also utilized at the transport layer. Link-layer schemes that use FEC to detect and recover from errors, duplicate the

effort, since TCP already uses checksums to detect errors. These unnecessary processing demands can impose substantial burden on a mobile device depending on the capabilities of the device and the number of network applications running on it.

The TCP-aware link-layer solutions, such as the Snoop protocol [7], attempt to solve some of the issues just described. The Snoop protocol introduces a module, called the *snoop agent*, at the basestation. The agent monitors every packet that passes through each TCP connection and maintains a cache of TCP segments sent across the link that have not yet been acknowledged by the receiver. The state maintained at the Snoop agent for each connection is soft, and is not necessary for correctness. Snoop detects losses in the same way as TCP, i.e., by the arrival of several duplicate TCP acknowledgments or due to a local timeout. Upon detection of a packet loss, the Snoop module retransmits the lost packet from the local cache and suppresses any duplicate acknowledgments heading towards the TCP sender. The key feature of the Snoop protocol is that link-layer retransmissions are TCP-aware. Specifically, the Snoop protocol leverages off TCP acknowledgments arriving from the mobile host in order to detect packet losses. The duplicate acknowledgments are suppressed to shield the TCP sender from non-congestive-losses. Furthermore, Snoop uses timeout values that are smaller than TCP timeouts, so that it has an opportunity to recover from losses before TCP times out and invokes congestion avoidance mechanisms. The combination of these techniques reduces the overhead incurred due to link-layer retransmissions and also reduces the competing effects between the ARQ schemes of the transport and the link layers.

One disadvantage of the Snoop protocol is that, like other link-layer protocols, it could also suffer from not being able to completely shield the sender from wireless losses [11]. This can occur if the Snoop module fails to recover from packet losses before TCP times out. The problem can be minimized by using a finer timeout value at the Snoop module, however at the cost of incurring more processing load on the basestation. The second

disadvantage is that the protocol violates the layering principle of protocol design due to its use of TCP acknowledgment for link-layer recovery. The third possible disadvantage of the Snoop approach is that it requires a slight modification in the TCP implementation of the mobile host, so that congestion and non-congestion losses can be differentiated. Specifically, the Snoop module is equipped to send negative acknowledgment (NACKS) to the TCP sender on the mobile host when missing packets are detected. As will be seen later, we will show that modification of the existing TCP implementations on *mobile hosts* is not a stumbling block in seamlessly integrating applications that use the TCP protocol, and therefore, not a major issue.

2.2 Split Connection Schemes

Split connection schemes split the end-to-end connection into two connections – one TCP connection between the sender and the basestation and another between the basestation and the receiver. The protocol on the wireless segment does not necessarily have to be TCP; it can be any other reliable transport protocol. The idea behind this approach is to separate loss recovery over the wireless segment from that across the entire end-to-end connection. This separation allows a reliable transport protocol on the wireless segment to quickly recover from packet losses, thereby shielding the TCP sender from the wireless link. Indirect TCP is an example of the split connection approach that uses "regular" TCP for its connection over the wireless segment. Experiments indicate that the choice of TCP on the wireless segment results in several performance problems. The TCP sender on the wireless segment still interprets all wireless losses as congestion problems and invokes various congestion control mechanisms leading to a sub-optimal performance. Another problem is that every packet incurs the additional overhead of traversing the transport layer twice at the basestation in order to hop over to the next connection. The problem can be solved by an efficient kernel implementation that avoids extra copies of packet buffers [11].

Category	Advantages	Disadvantages
Link-Layer (LL)	1. Fits well into layering structure. 2. Poor at shielding the sender from non-congestion losses. 3. Does not require connection state to be maintained at the basestation.	1. Competes with TCP and duplicates some effort. 2. Requires smaller timeout values: places more burden on mobiles. 3. Cannot solve the problem associated with handoffs.
Link-Layer-TCP Aware	1. More effective at shielding the sender from non-congestion losses than LL. 2. Depending on the implementation may not impose extra load on the mobile. 3. Only requires soft connection state to be maintained at the basestation.	1. Violates layering principles. 2. Possibly requires TCP on the mobile to be modified. 3. Requires extra link-layer messages for recovery.
Split-Connection	1. De-couples the error control on wireline segment from that of the wireless segment. 2. Allows development of a lightweight protocol on the wireless segment.	1. Requires connection state to be maintained at the basestation. 2. May incur more processing load at the basestation. 3. Acknowledgments are not end-to-end.
End-to-End	1. Explicit Loss Notification (ELN) can help sender to distinguish between congestive and non-congestive losses.	1. Slow to recover from losses. 2. Requires TCP implementations on fixed hosts to be modified.

Table 1: Comparison of the different approaches

One problem, particularly in the implementation of I-TCP, is that the end-to-end semantics of the TCP protocol are not preserved [11]. Proponents of the I-TCP approach, however, argue that most applications that use TCP also have some kind of support for application layer acknowledgment and error recovery. Such acknowledgments are often required, because TCP does not provide any notification to the sending application when the data is actually delivered to the peer application [6]. The other problem that is widely stated in the context of the split-connection scheme is that it requires substantial amount of state to be maintained at the basestation making handoff procedures complex and slow [11].

2.3 End-to-End Schemes

The two schemes discussed so far attempt to recover from non-congestion losses locally, thus shielding the TCP sender from the wireless link. On the other hand, most effective end-to-end solutions add an explicit loss notification (ELN) feature to TCP acknowledgments. When a packet is dropped on the wireless link, subsequent cumulative acknowledgments corresponding to

the lost packet are marked by an agent on the basestation to indicate the occurrence of a non-congestion-related loss. The TCP sender uses this information to decide whether to invoke congestion control mechanisms or not. One problem with this approach is that end-to-end protocols are usually slow to recover from losses on the wireless link, since it takes more time for the TCP sender to detect problems [11]. The situation is exasperated in WAN environment, where the round-trip delays are much greater than in a LAN. The biggest problem with the end-to-end approach is the need to modify the existing TCP implementations on the *fixed hosts*, making the solution impractical for wide scale deployment.

2.4 Selective Acknowledgments

Typically, losses in the wireless environment consist of burst packet errors. The standard TCP protocol, which uses cumulative acknowledgments, does not provide the sender with sufficient information to recover from multiple packet losses. A selective acknowledgment (SACK) technique, called SMART Retransmission [12], contains the cumulative acknowledgment and the sequence number of the packet that caused the acknowledgment to be sent. The sender can use this information to construct a bitmap of the packets that have correctly reached the receiver. When the sender detects a gap in the bitmap, the corresponding packets are retransmitted without considering the possibility that the acknowledgments may be reordered or may simply be dropped. Researchers have studied SMART selective acknowledgment technique in conjunction with the above approaches (link-layer, split-connection and end-to-end) and compared them with each other [11]. The next section presents the performance results and the evaluation of these schemes.

2.5 Evaluation of Related Work

Table 1 summarizes the advantages and disadvantages associated with the different approaches. Various protocols in combination with SMART have been extensively studied and compared by a group of researchers in [11]. Here,

Name	Category	Description / Examples
E2E	End-to-end	Standard TCP-Reno
E2E-ELNRXMT	End-to-end	Standard TCP + ELN + retransmit on first duplicate ack.
E2E-SMART	End-to-end	Standard TCP + SMART selective acks
SPLIT	Split connection	Standard TCP on both segments
SPLIT-SMART	Split connection	Standard TCP + SMART on wireless segment
LL	Link-layer	Simple link-layer protocol
LL-TCP-AWARE	Link-layer	Snoop protocol
LL-SMART-TCP-AWARE	Link-layer	Snoop protocol + SMART

Table 2: Summary of various protocols

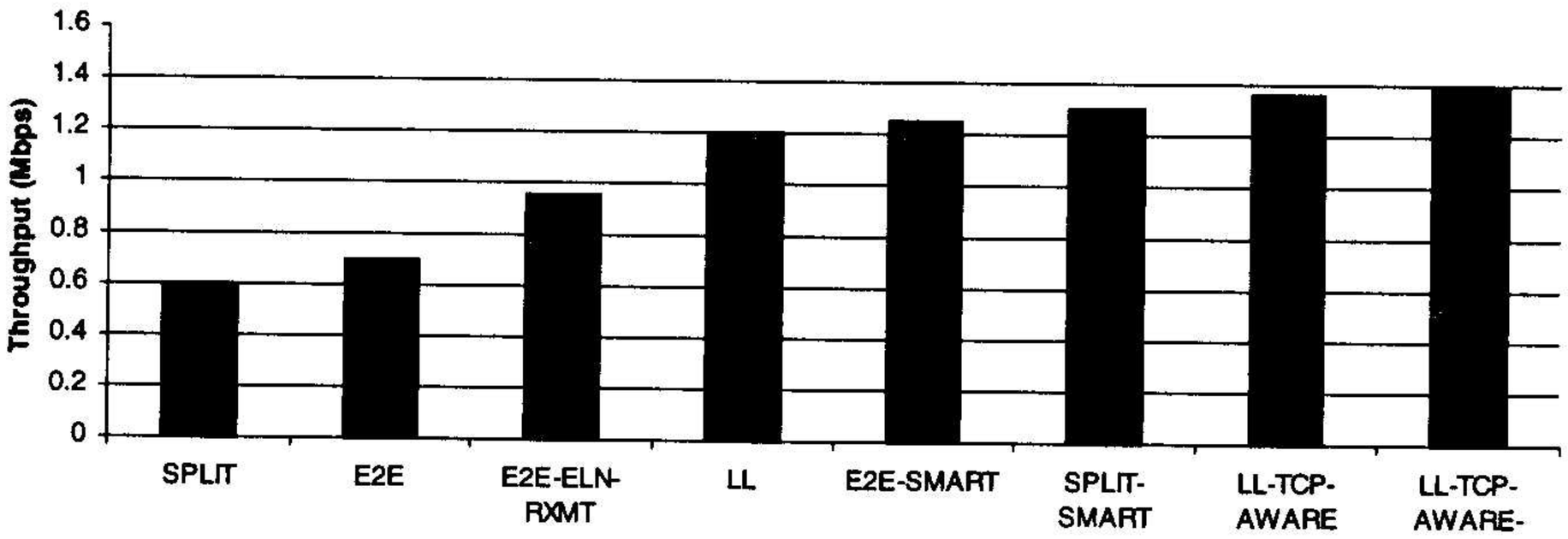

Figure 2: Performance of various protocols at bit-error rate of 1 error/65536 bytes[1]

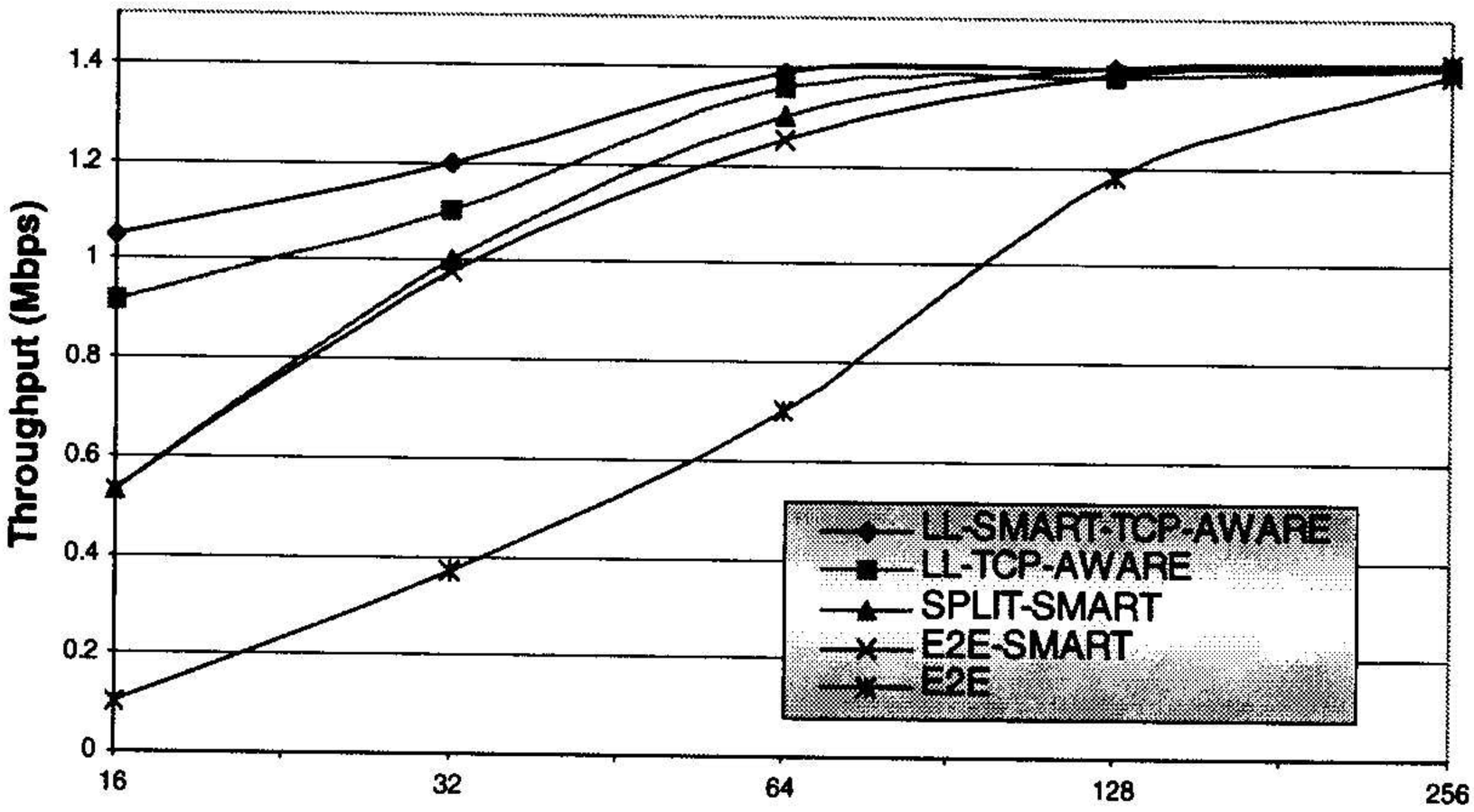

Figure 3: Comparison of throughput at various error rates, ranging from 1 error every 16 KB to 1 every 256 KB[1]

[1] *These results are obtained from [11].*

we briefly present some important elements of their findings. Their experimental testbed consisted of IBM ThinkPad laptops and Pentium-based PCs running BSD/OS 2.1 from BSDI. The machines were interconnected using 10Mbps Ethernet and AT&T Wavelan with a raw signaling bandwidth of 2Mbps. The peak throughput for TCP bulk transfers under ideal (no loss) conditions was 1.5Mbps in the local area testbed. Errors were generated on the lossy link using an exponentially distributed bit-error model and packet corruption was achieved by changing the TCP checksum. All measurements were taken on bulk transfers from TCP sender on the stationary host to the receiver on the mobile host using data packet size of 1400 bytes. Table 2 [11] enumerates some of the protocols that were evaluated. These are the protocols that we mainly focus on here.

The following observations can be made from the results in Figures 2 and 3:

a. SMART selective acknowledgments greatly improve the performance of each category of protocols. For example, the difference in throughput between SPLIT and SPLIT-SMART is very large, as is the difference in the case of the end-to-end protocols. Selective acknowledgments provide a mechanism to quickly recover from multiple packet losses and thereby reduce the occurrence of a timeout.

b. The SMART technique alone, however, does not perform very well under high loss rates. Both the LL-TCP-AWARE and the LL-SMART-TCP-AWARE protocols outperform SPLIT-SMART and E2E-SMART in heavy loss conditions. The key reason is that the TCP-aware link-layer protocols shield the TCP sender from wireless losses. Unlike the link-layer protocols, the SPLIT-SMART and the E2E-SMART protocols invoke congestion control mechanism upon experiencing a packet loss. In addition, as the congestion window size remains small, it significantly reduces the possibility for the TCP sender to fast retransmit lost packets.

c. Although, E2E performs slightly better than SPLIT under the LAN environment, results (not shown here) also indicate that SPLIT outperforms

E2E in the WAN environment [11]. This implies that local loss recovery is quicker and more effective than end-to-end recovery.

d. Minor performance differences between split-connection and other schemes under low loss condition indicates that the extra processing load on the basestation due to the connection split is negligible.

e. The LL protocol is not very effective in shielding the TCP sender from wireless losses. When a loss occurs, the LL protocol performs a local retransmission. However, enough packets from the TCP sender are typically in transit to create more than 3 duplicate acks. Since the LL protocol does not suppress any duplicates, they propagate to the sender and trigger a fast retransmission and the associated congestion control mechanisms [11]. As expected, the LL-TCP-AWARE protocol performs better than the LL protocol.

3. Mobile TCP (MTCP)

Based on the experience and the results of previously studied approaches, we conclude that the characteristics of an ideal solution must include the following:

- *local recovery from wireless losses*
- *capable of completely shielding the TCP sender from wireless (non-congestion) losses*
- *use of some form of selective acknowledgment scheme to recover quickly from losses*
- *no requirement for re-linking of applications*
- *minimization of processing load on mobile hosts*
- *effective in supporting mobility, i.e. handoffs*

Our approach in achieving many of these goals is to design Mobile TCP (MTCP), a lightweight transport protocol, for the wireless segment based on the split-connection model. The Mobile TCP protocol is based on our previous work in this area, as described in [9]. The key idea influencing our protocol design is the fact that Mobile-TCP operates over a single point-to-point connection. Thus, in many aspects, the design of MTCP resembles that of a link layer protocol [9].

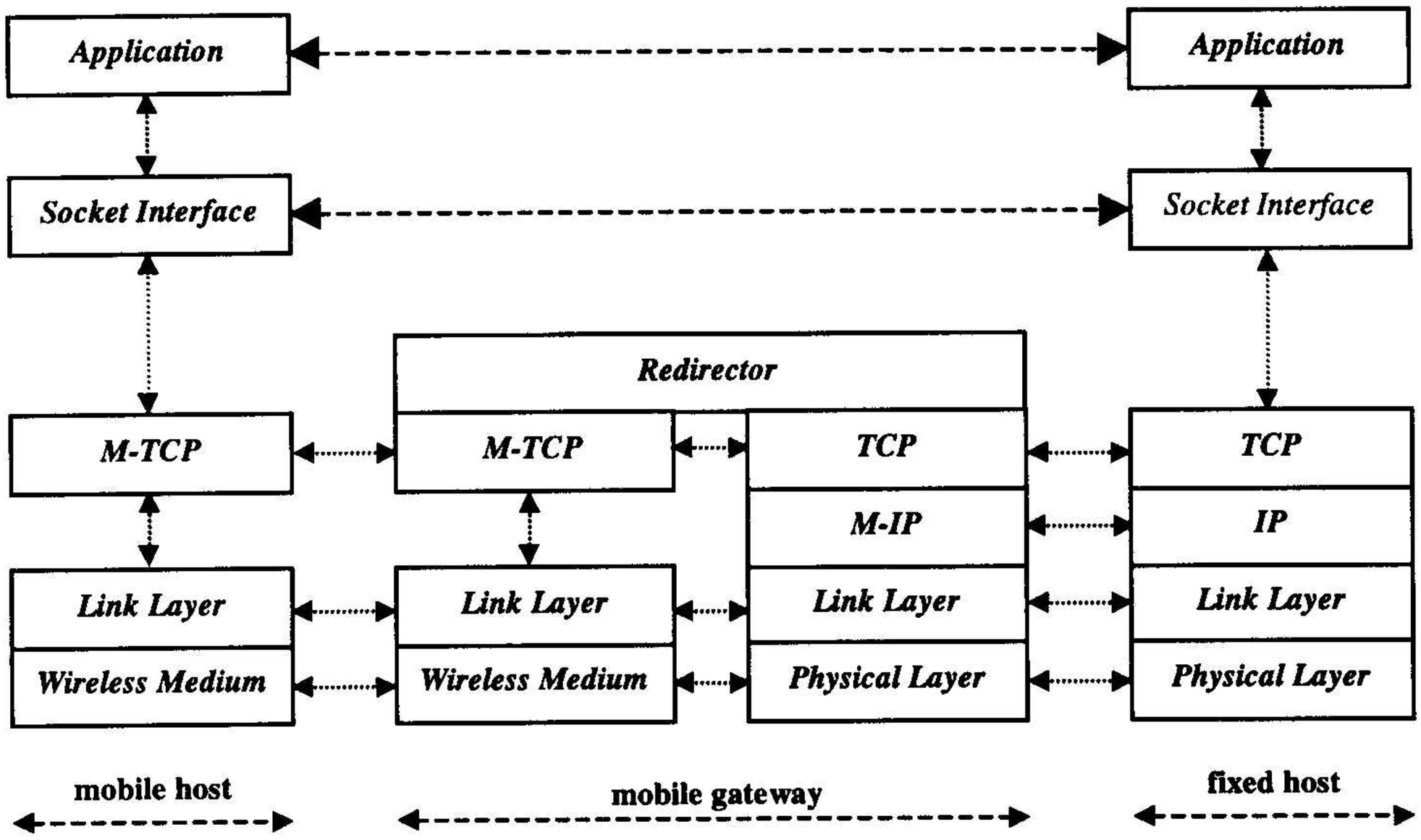

Figure 4: Mobile-TCP Protocol Stack

Furthermore, indirection provides the property that all packets arrive in order and all losses are non-congestion-related on the wireless segment. As a consequence, the design of MTCP can be greatly simplified and optimized, resulting in a significant reduction in the communication load over the wireless link.

The features of our MTCP implementation are as follows:

- elimination of the IP layer
- header compression
- no congestion control mechanisms and timer backoffs
- static window-based flow control
- selective acknowledgments optimized for burst packet losses
- explicit retransmission requests
- fast recovery from retransmission losses

Now, we proceed with the detailed description of the Mobile-TCP design. The next few sections deal with the design of connection control, flow control and error control mechanisms of the protocol.

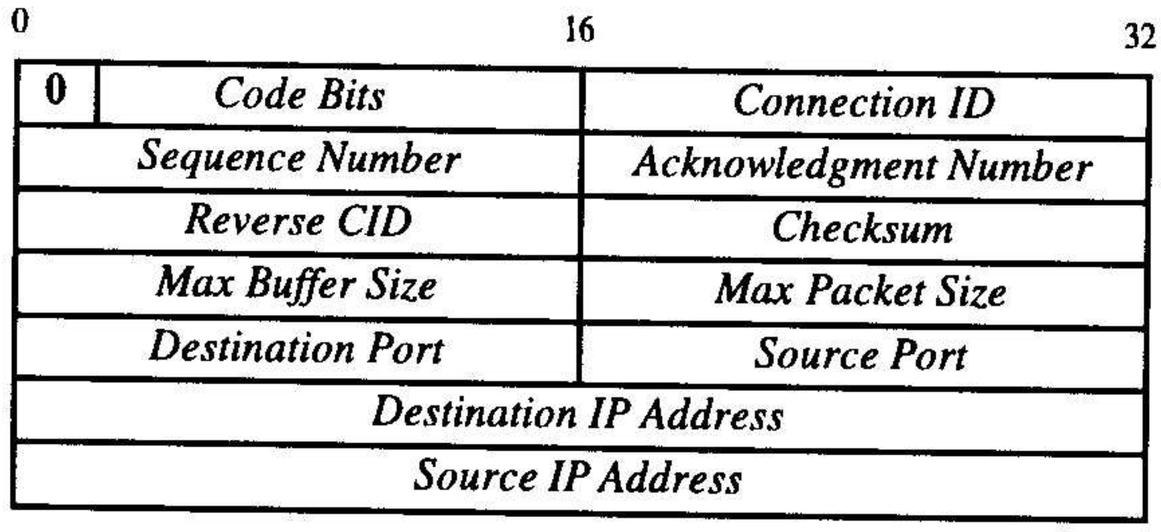

Figure 5a: Connection Control Packet

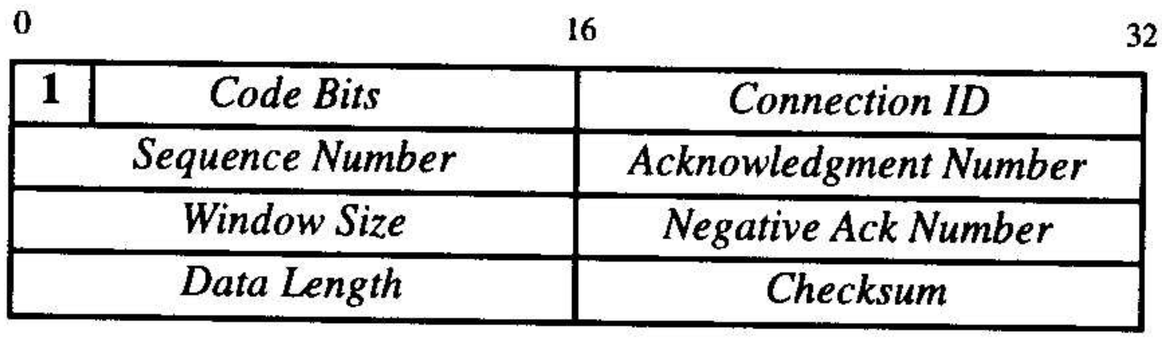

Figure 5b: Data Packet

3.1 Connection Control

Connection control is concerned with setting up and tearing down connections. Under TCP, a host can initiate a connection with another host (called an active open) or can listen for and accept connections from other host (called a passive open). In the split-connection model, the mobile-gateway (MG) acts as an intermediary, which establishes independent connections with the mobile host (MH) and with the fixed host (FH). When the MH executes an active open, it communicates to the mobile-gateway the identity of the destination, as well as a set of parameters that will govern the operation of the connection between the MH and the mobile-gateway. The mobile-gateway then proceeds with setting up the connection with the fixed host using the destination IP address and port number specified by the mobile host. Similarly, when a MH executes a passive open, it informs the mobile-gateway that it is prepared to accept connections. This allows the mobile-gateway to accept or reject connection requests from FHs without communicating a priori with the MH. Similar technique is used at the mobile-gateway for connections initiated by the fixed hosts. Since the Mobile-TCP architecture, as shown in figure 4, completely

eliminates the IP layer from the protocol stack, it subsumes the IP addressing functionality into its own packet headers.

In addition to eliminating some IP header information, we reduce the amount of wirelessly transmitted data by employing a technique similar to the Van Jacobson's header compression [13]. Since all the packets from the mobile host travel through the mobile-gateway, there is no need to communicate the full source and destination IP addresses (including the port numbers) in every packet. Instead, at the connection establishment stage, a connection ID (CID) is assigned for each direction and is used in subsequent exchanges of data over the wireless segment [9]. (CID_{MH} represents the CID from MH to MG and CID_{MG} is the CID in the reverse direction). The connection control and user data packet formats are shown in figures 5a and 5b. Note that the first bit is used to distinguish between the two formats. The MTCP protocol uses a three-way handshake similar to TCP to establish connections and exchange unique connection identifiers between the mobile host and the mobile-gateway. The bindings between CIDs and IP addresses (including port numbers) are cached at both ends of the connection. When sending a packet from the mobile to the network, the destination IP address is translated into the corresponding CID_{MH} at the mobile host. At the mobile-gateway, the CID_{MH} is expanded back into the destination IP address, which is used on the wireline segment of the connection. Similar operation is performed in the reverse direction. In practice, the Redirector, which sits above the transport-layer and is responsible for "routing" data from the source to the destination, only has to maintain a table of bindings between the MTCP and the TCP socket descriptors.

We follow similar implementation strategies as in I-TCP [6] to seamlessly establish an indirect connection such that full compatibility is ensured with the existing TCP/IP stack on the fixed host. The problem in case of I-TCP, however, is that it requires applications on the mobile host to use new socket calls for the purpose of connection establishment. We argue that this is unnecessary in our case, since this functionality is transparently embedded into the MTCP protocol. Later, we will address the issue of application re-linking on

the mobile host. We direct our focus now on issues that have the most significant impact on the overall performance of the protocol.

3.2 Flow Control

As stated earlier, congestion control mechanisms in TCP affect the throughput by constricting the window size of the sender when losses are detected. Since all losses on the wireless segment of the connection are non-congestion-related, a better approach to flow control design is to de-couple it from error control strategies. In MTCP, flow control is achieved by a static sliding window protocol. This scheme is identical to the one used by TCP [4], except that congestion control mechanisms are not implemented. As a consequence, the sender window size is entirely controlled by the receiving end and the maximum size of the window is limited by the size of the MTCP receive buffer. An optimal buffer size should be selected to be at least as large as the bandwidth-delay product. A buffer size that is smaller than the bandwidth-delay product can cause the wireless link to be underutilized, while larger than optimal size can result in greater buffering requirements on the link-layer protocol drivers. In traditional networks, greater than optimal window sizes can also lead to severe congestion and traffic problems resulting in the loss of packets. However, this does not pose a problem in a point-to-point configuration, such as the one we are addressing here. In fact, in our experience, reasonably large window sizes significantly improve the chances of quickly detecting packet losses and allow for fast recovery, rather than relying on retransmission timeouts. Nevertheless, due to the various factors involved in choosing the receive buffer size, which are not necessarily in the realm of transport protocols, we have made it a configurable parameter, as is done in TCP. We would like to look at the issue of dynamically choosing the optimal buffer size in our future research.

3.3 Error Control

TCP implements error control functions in the sender as well as the receiver. Specifically, the TCP receiver's main objective is to perform error

detection by the means of checksums and holes in received sequence numbers. When the receiver detects an error, it signals the sender of the loss using a duplicate acknowledgment. In turn, the sender identifies losses either by the arrival of three duplicate acknowledgments or by the absence of an acknowledgment within an estimated timeout period. Upon identifying a loss, the TCP sender invokes various congestion control mechanisms and then transmits the first unacknowledged packet followed by a number of outstanding (pending transmission) packets that are allowed by the window size. In essence, TCP takes a conservative approach to loss recovery by assuming that only a single packet is lost. The technique is well suited for networks that experience light congestion, but too slow to recover in environments with heavy losses.

MTCP adopts a strategy similar to TCP for error control with the exception of few key differences that make it far more robust in recovering from heavy loss conditions. These differences can be stated as the following:

1) Explicit retransmission request (RRQ) – Upon detecting an out of order packet, the receiver immediately requests the sender to quickly retransmit the lost packet(s) as opposed to waiting for three duplicate acknowledgments. This is a direct consequence of the fact that all packets on the wireless segment arrive in order, which is guaranteed by the split-connection model. An out of order packet thereby automatically signifies a loss on the wireless link.

2) Selective acknowledgments (SACKs) optimized for burst losses – MTCP adopts a technique for selective acknowledgment that is slightly different from the SMART scheme. Recall that in SMART every acknowledgment packet contains the sequence number of the packet that triggered the acknowledgment to be sent. This enables the sender to construct a bitmap of lost packets. The approach, however, has two problems: i) it is complex to implement, and ii) lost acknowledgments cause the sender to overestimate the magnitude of the loss. Instead, every MTCP acknowledgment contains a negative acknowledgment number, which conveys the size of the burst

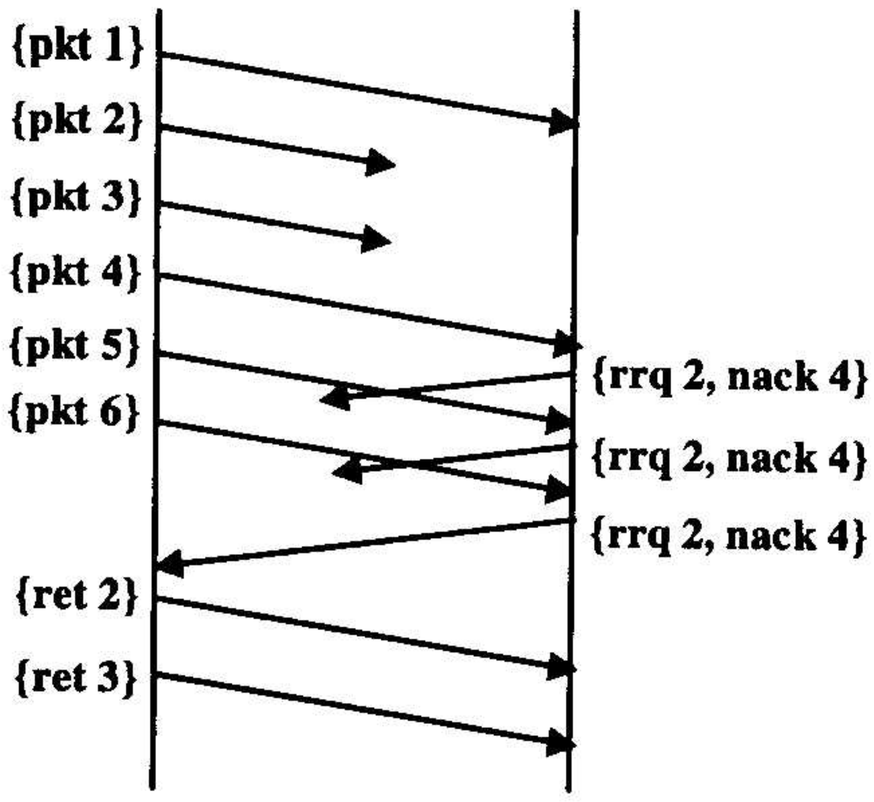

Figure 6a: MTCP SACK

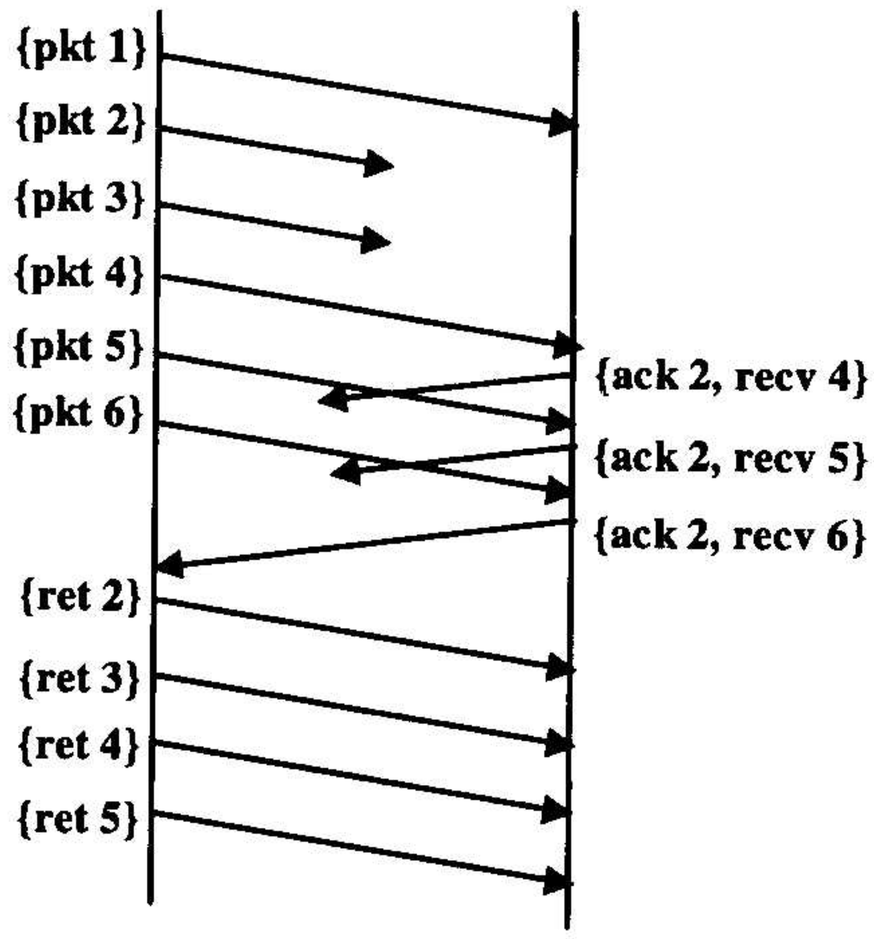

Figure 6b: SMART SACK

loss in bytes to the sender. Since this information is repeated in every acknowledgment, the MTCP sender obtains an accurate knowledge of burst losses even when some acknowledgment are lost. Figure 6 compares between the MTCP approach to SACKs versus the SMART technique. Figure 6a shows that the receiver requests three times the retransmission (RRQ) of packets 2 and 3. Two of these requests are corrupted or lost. Upon receiving the third request, the sender retransmits packets 2 and 3. On the other hand, SMART technique interprets the third retransmission request (ack 2, recv 6) as an indication of the loss of packets 2, 3, 4 and 5, causing it to retransmit in excess. Our scheme is comparatively much easier to implement than SMART and equally effective, if not more, in burst loss conditions. The limitation of the MTCP technique is that it cannot guess multiple sparse (non-burst) losses. We argue that most losses in wireless links are burst losses and therefore our technique is sufficient to handle this situation. Furthermore, intuition tells us that the MTCP window size is large enough (i.e. unrestricted by congestion control) to trigger fast retransmissions in most multiple sparse loss cases. Our performance results show that this scheme is indeed quite effective in combating losses.

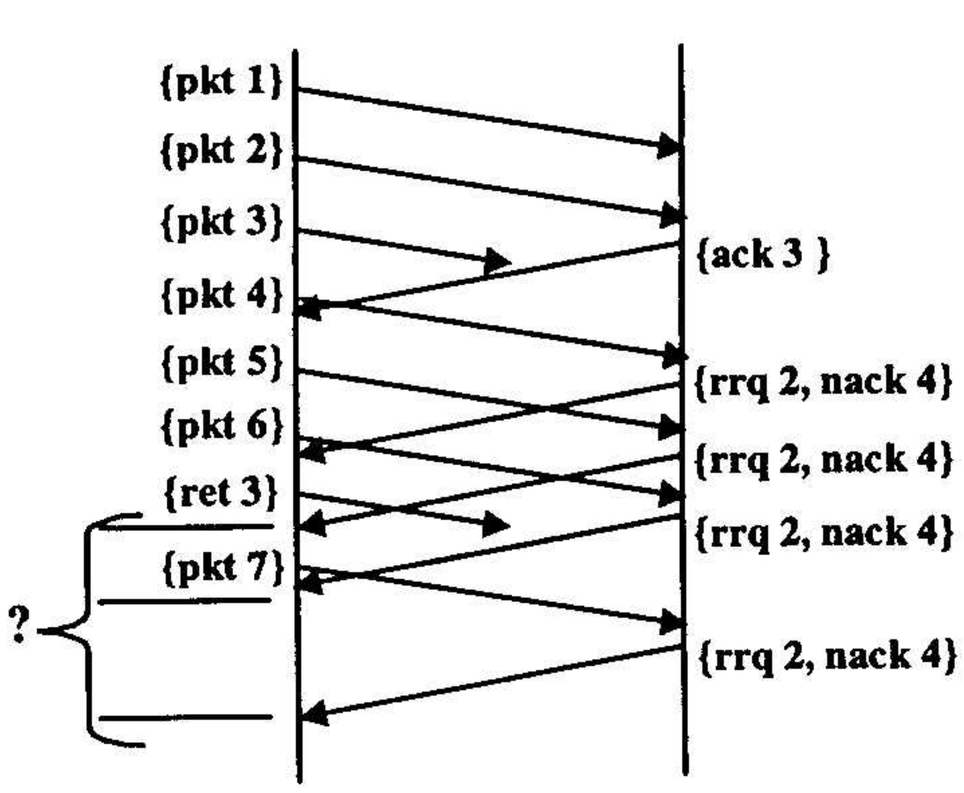

Figure 7a: Problem

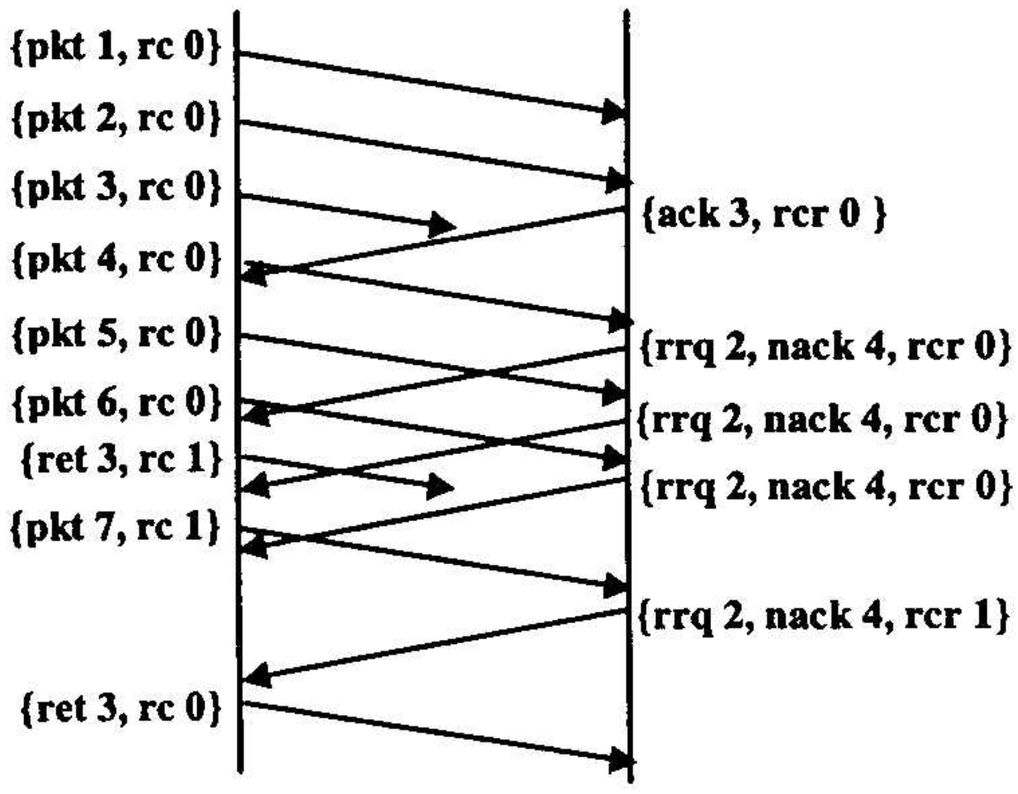

Figure 7b: Solution

3) Fast recovery from retransmission losses – Since losses are very common on the wireless segment, MTCP tries to take every step to quickly recover from errors. The same applies to packets lost during retransmission. When an MTCP sender receives a request for retransmission of a packet that was just retransmitted, the sender is unable to resolve whether the request is for the retransmitted packet or for the previously failed transmission. The situation is best illustrated in figure 7a. The key to differentiating between the two requests is for the sender to somehow know that the request was in response to the retransmission. In figure 7a, the retransmission requests (RRQs) triggered by packets 5 and 6 are indistinguishable from the RRQ triggered by packet 7. Ideally, the sender should respond with the retransmission of packet 3, when it gets the RRQ due to packet 7 and ignore the rest. However, the request packet does not contain adequate information for the sender to differentiate between the requests. One obvious possible solution is to use SMART-style sequence number of the received packet in the RRQ packet. We have opted for a more compact solution that can be achieved with just 2 bits (included in the CodeBits vs. an extra 16-bit field in the MTCP header). In our solution, every packet carries a retransmission-cycle number (RC) and every acknowledgment in the opposite direction contains a retransmission-cycle reply (RCR). When a receiver responds to the sender, it simply copies the previously received RC bit to the RCR bit of the acknowledgment packet.

Upon receiving a retransmission request, the sender checks to see if the RCR bit is equal to its present retransmission-cycle (PRC). If there is a match, the PRC bit is toggled and requested packets are retransmitted with the new cycle number. Otherwise, the retransmission request is simply ignored by the sender. This action prevents the sender from responding to duplicate retransmission requests. Figure 7b shows that with this scheme, the sender can now distinguish between the RRQs triggered by packets 5, 6 and the RRQ triggered by packet 7. Note that the technique works even for multiple retransmissions.

4. Implementation Details

We have implemented the MTCP protocol under the Microsoft Windows 95/NT platform. As the TCP source code for Windows was not available to us, we have designed and developed the entire MTCP protocol from scratch. The protocol is implemented as an intermediate protocol driver in the kernel mode and communicates with NDIS-compliant link-layer drivers [14] to send and receive packets on the network card. Thus, we achieve complete elimination of the IP layer from the processing path of the transport protocol. An important consequence of this is that buffer management is greatly simplified and optimized in MTCP. Furthermore, our implementation requires only a single memcpy per send or receive operation (apart from possible copies in the link-layer driver).

Most TCP implementations, such as BSD, employ two types of timers: fast timer (200ms) and slow timer (500ms) [4]. The TCP delayed acknowledgments are processed on fast timeouts and TCP retransmissions on slow timeouts. We would like timeout granularities to be on the order of the round-trip-time (RTT). However, in order to limit the overhead of processing timer interrupts, the MTCP protocol utilizes a single timer that goes off every 200ms to process both delayed acknowledgments and retransmissions. Interestingly, our experiments indicate that timeout values on the Microsoft TCP (MSTCP) implementation are more accurate than on the BSD implementation.

We have seen timeout values in the range of 200-500ms on MSTCP, while most timeouts on BSD are typically multiples of 500ms.

MTCP follows the standard BSD implementation to estimate the smoothed round trip time. Consequently, retransmission timeout interval in MTCP is computed as the sum of the smoothed RTT and four times its mean deviation [4].

5. Experimental Results

We performed various experiments to compare between the performance of MTCP, Indirect TCP and end-to-end TCP under wireless communication conditions. In the following sections, we present our experimental setup and discuss our results using various criteria for performance evaluation.

5.1 Experimental Setup

Our experimental testbed, shown in Figure 8, consists of Pentium-based personal computers running Windows 95 and Linux operating systems. The Win95 machine in the center of the figure is configured to act as a basestation using WinRoute software to route packets to and from the mobile host. As indicated in the figure, the basestation and the mobile host are connected to each other using either 10Mbps Ethernet or Proxim RANGELAN, a wireless LAN with a raw signaling bandwidth of 1.6Mbps. We chose to test the protocols under two different conditions to observe the effects of MTCP optimizations at different data rates. The peak throughput for TCP bulk transfers is measured to be approximately 8Mbps on the 10Mbps Ethernet and 0.70Mbps on the 1.6Mbps RANGELAN [15].

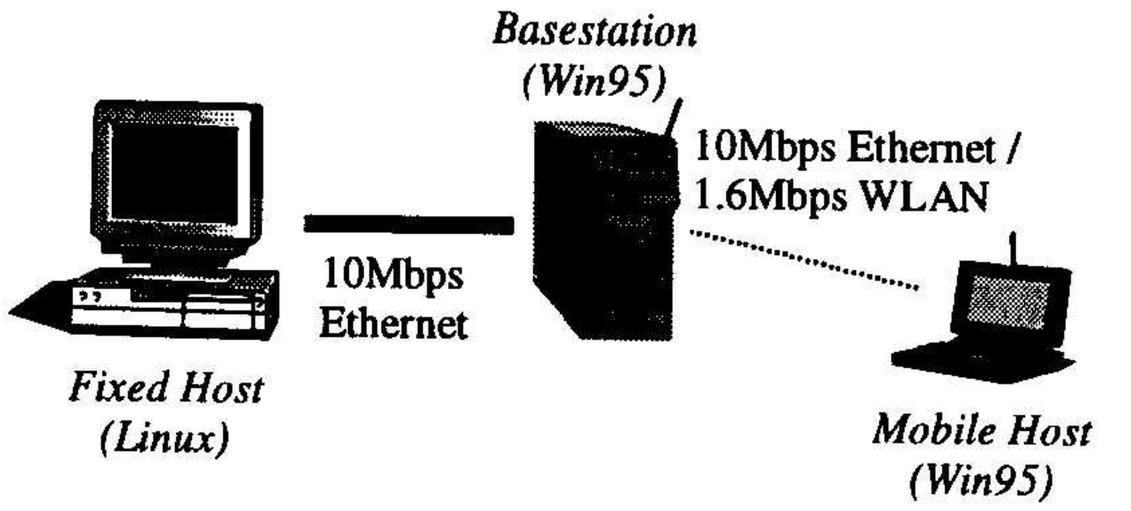

Figure 8: Network Configuration

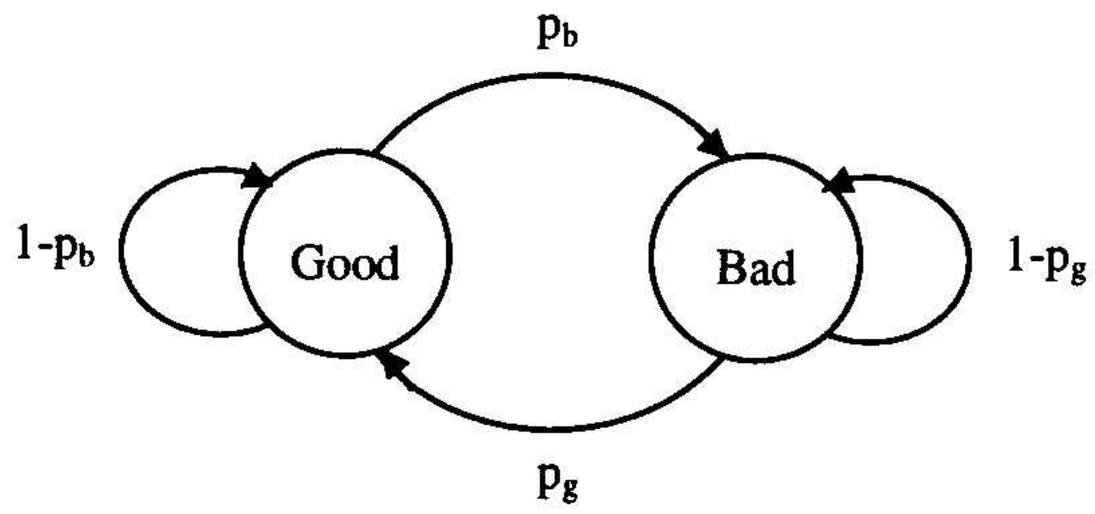

Figure 9: Burst-error model for lossy link

We simulate errors on the lossy link using a Markov chain burst-error model, shown in Figure 9. This model attempts to incorporate the fading effects experienced by the wireless channel. Specifically, the beginning of a fade corresponds to the transition from the **Good** state to the **Bad** state. Once in the bad state, the model continues to drop bits at a higher probability causing a burst loss of bits. Finally, the end of fade is marked by the transition from the **Bad** state to the **Good** state. The model is designed to be independent of the transmission rate of the channel, i.e. every bit is subject to the same drop probabilities regardless of its length in time. The properties of the burst-error model can be summarized by the following equations:

$$\text{[eq. 1]} \quad \text{Expected BER} = \frac{p_b}{p_b + p_g}$$

$$\text{[eq. 2]} \quad \text{Expected Burst Size in Bits} = \frac{1}{p_g}$$

$$\text{[eq. 3]} \quad \text{Expected Burst Size in Packets} = 1 + \frac{1}{Bp_g}$$

$$\text{[eq. 4]} \quad \text{PER}/\text{BER} = Bp_g + 1$$

where B is the average packet size in bits

Regardless of the error-model, a packet intruder driver modifies the checksum of outgoing packets in order to induce packet errors. The intruder specifies the burst-error parameters (p_b, p_g) and the size of the outgoing packet in bits to the Markov state machine shown in Figure 9, which thereby determines whether the packet will be corrupted or not. Losses are generated in both directions of the wireless link, so acknowledgments are dropped as well. For our simulations, the probability p_b is chosen between the range of 1E-7 and 120E-7 and p_g is fixed at 100E-6. This provides an approximate burst size of 10000 bits (or 1250 bytes) which corresponds to an expected burst loss length of 1.83 packets assuming that the packet size is 1514 bytes. The maximum packet size supported by our protocols and the physical media is 1500 bytes (excluding 14 bytes of the data link header). The maximum receive buffer size is set to 16 Kbytes for all the protocols.

5.2 Performance and Comparisons

We compared the performance of MTCP with Indirect TCP and end-to-end TCP using metrics such as normalized throughput, mean inter-buffer arrival time, mean CPU usage and mean energy consumption. Our experimental results indicate that MTCP performs significantly better than TCP depending on the packet error rate and the data rate of the link. In order to compare the results obtained for each protocol at different data rates, we based our evaluation of the performance on normalized throughput. We define normalized throughput as the percentage of the maximum throughput that is achieved under ideal (i.e., no loss) conditions. Figure 10 shows the performance of MTCP, I-TCP and end-to-end TCP for bulk transfer from the fixed host to the mobile host in Wireless (Fig. 10a) and Ethernet configurations (Fig. 10b). The results are shown for packet error rates ranging from 0.2% to 20% on a logarithmic scale. Note that the packet error rates shown in the figures correspond to errors occurring in the sender to receiver direction only. Since packets in the reverse direction only contain acknowledgments, the error rate in this direction is very small in comparison. The results indicate that MTCP is able to sustain a greater normalized throughput

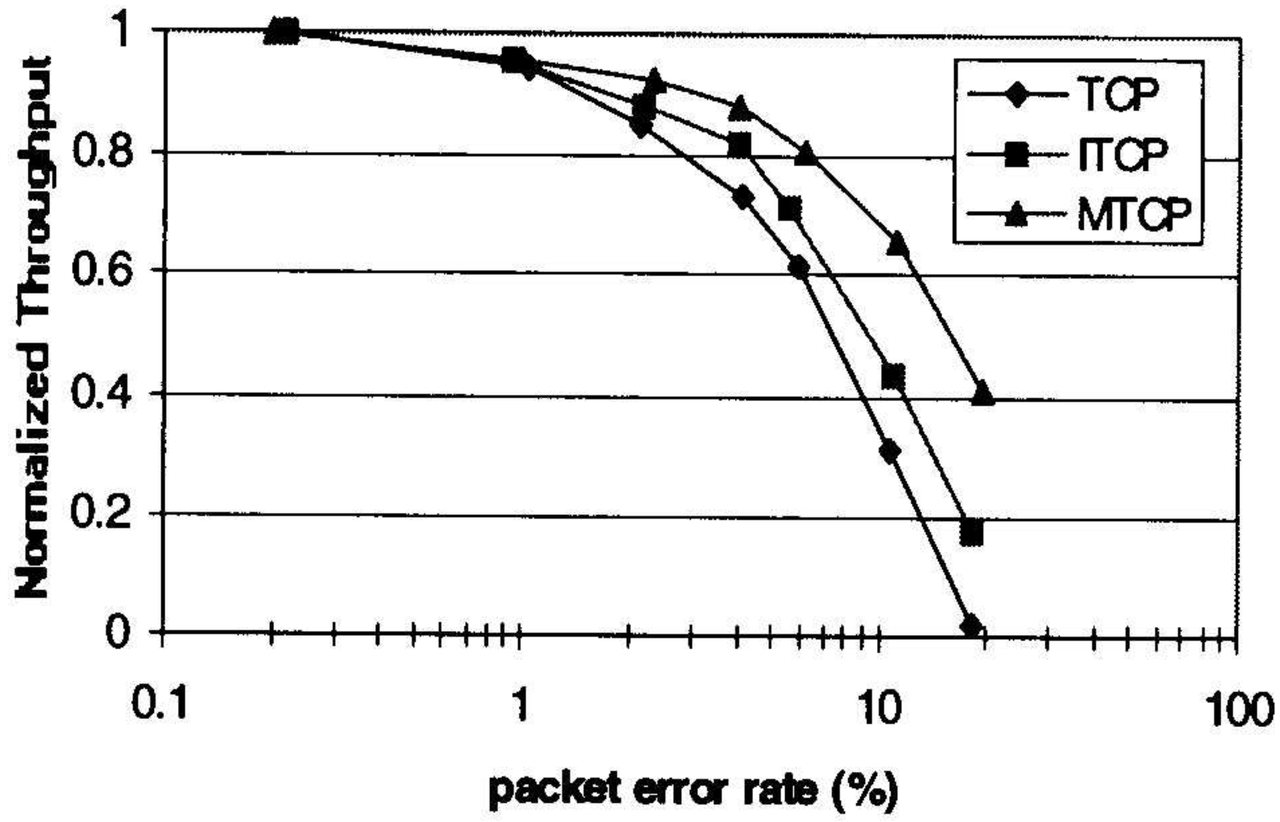

Figure 10a: Normalized Throughput vs. Packet error-rate for the wireless configuration.

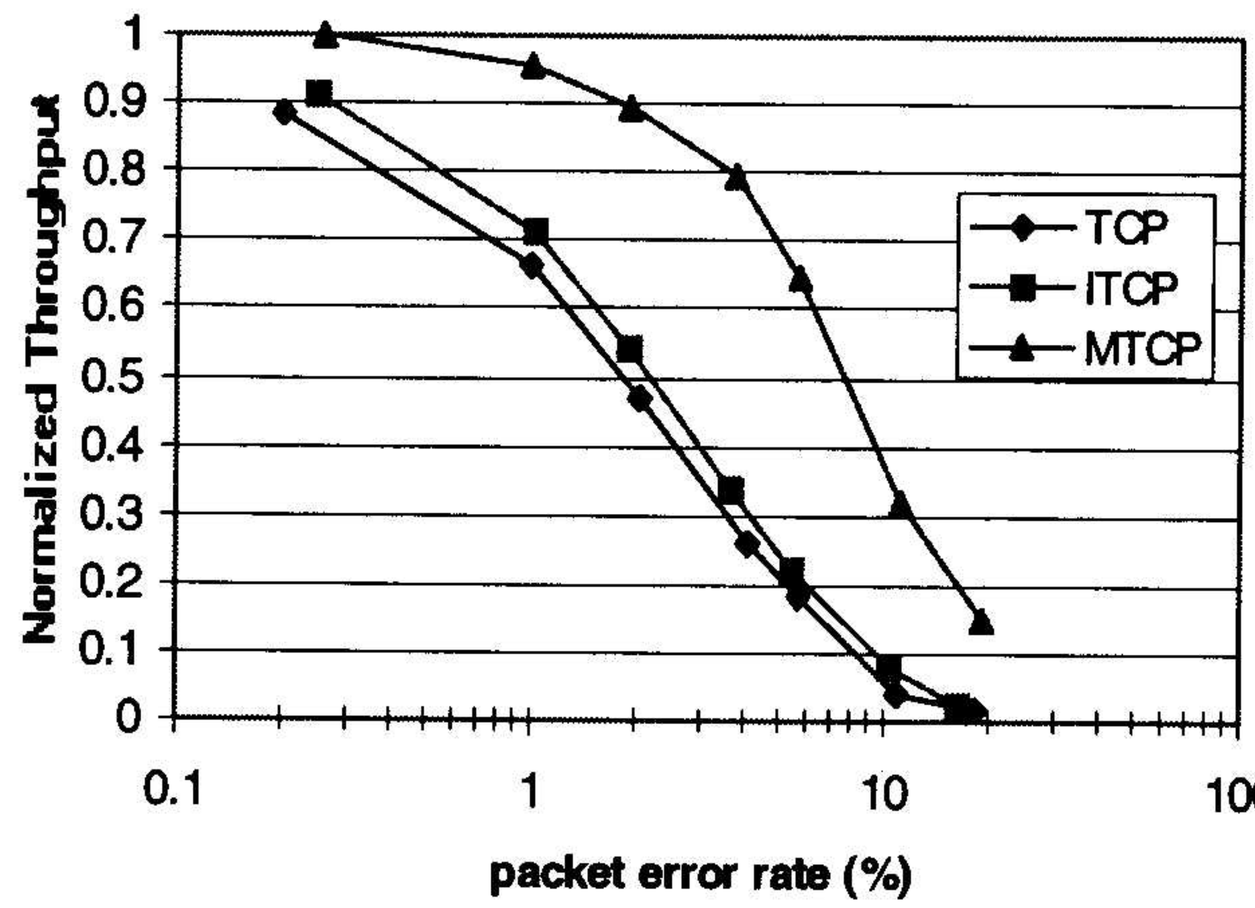

Figure 10b: Normalized Throughput vs. Packet error-rate for the ethernet configuration.

than both I-TCP and end-to-end TCP as packet error rates increase. The improvement is primarily due to: a) the elimination of congestion control mechanisms at the sender, b) the receiver-initiated retransmission requests (RRQs) that cause the sender to immediately retransmit lost packets, and c) the selective acknowledgment scheme for burst losses. The three techniques in combination improve the likelihood of fast re-transmitting lost packet(s), and thus reduce the possibility of an MTCP coarse timeout. The I-TCP performs worse than MTCP, since it continues to invoke congestion control mechanisms

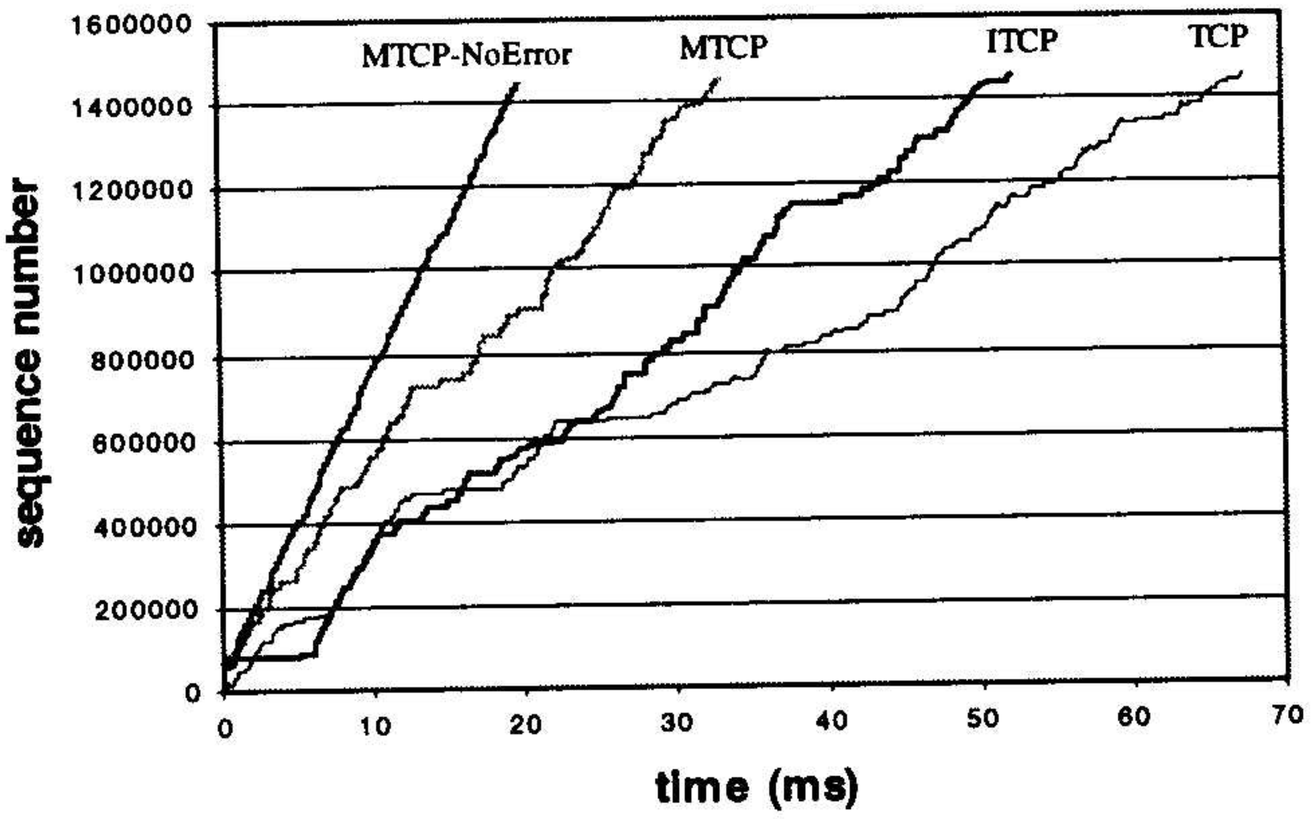

Figure 11: Protocol sequence trace at approximately 5% packet error rate in the wireless configuration.

upon packet loss on the wireless segment, causing an unnecessary reduction in the overall throughput. However, improved performance of I-TCP over end-to-end TCP can be attributed to local loss recovery, which reduces the time to recover from errors and also to the protocol differences that may exist between the Linux and the Windows implementation of the TCP/IP stack. Figure 11 graphically illustrates the progress of the data transfer from fixed host's standpoint in the Wireless configuration at approximately 5% packet error rate. It is apparent from this figure that both I-TCP and end-to-end TCP stall much more often in the presence of errors than MTCP and therefore, take longer time to complete the data transfer.

Figure 12a compares the results obtained for normalized throughput on Wireless and Ethernet configurations at various packet error rates. It can be clearly seen from the results that the protocols perform better in the Wireless configuration than in the Ethernet configuration by maintaining a higher normalized throughput as packet error rate increases. This can be explained by the fraction of the total transmission time spent idling in timeouts. As the transmission time decreases, the overhead due to coarse timeouts increases, thus resulting in lower normalized throughput in the Ethernet scenario. Since the packet error rate itself is dependent on the data rate of the connection, we also

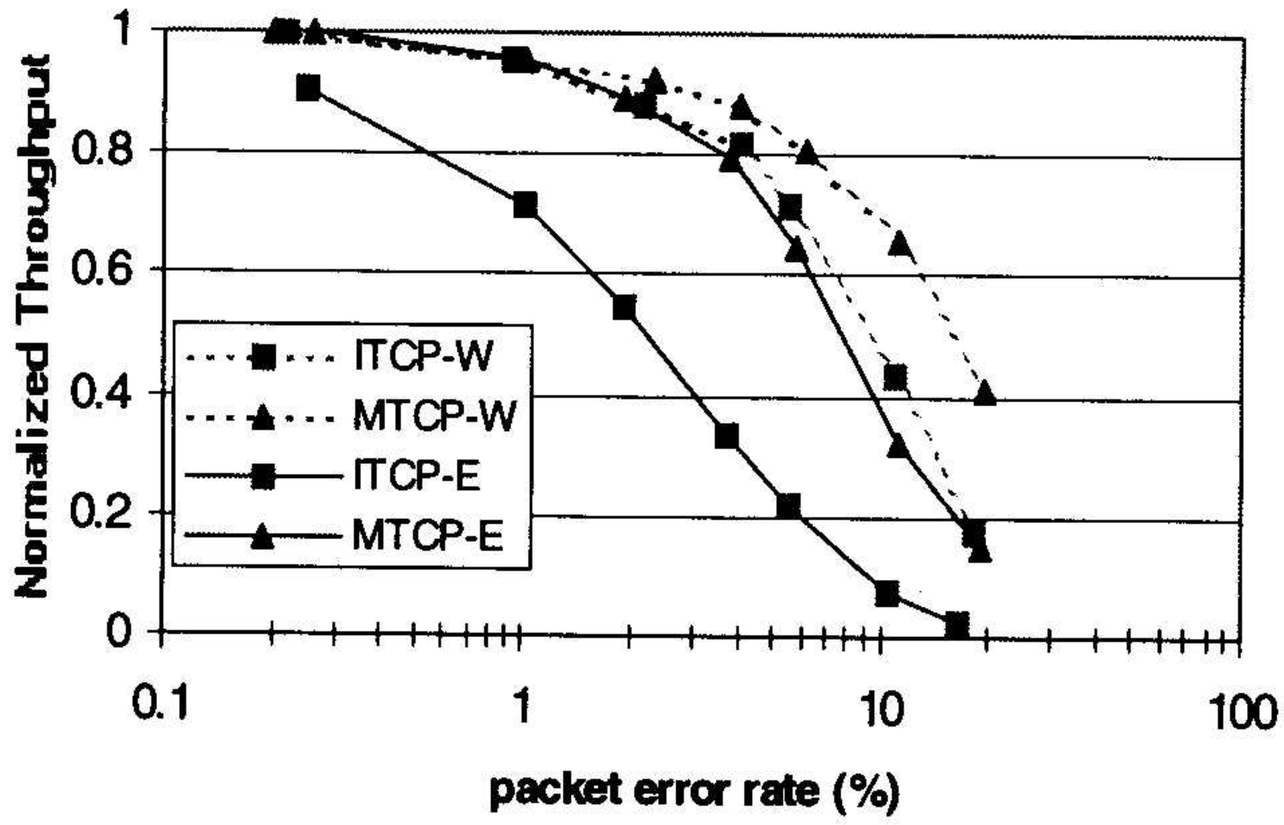

Figure 12a: Comparison of normalized throughput in wireless (denoted by suffix –W) and ethernet (-E) configurations at different packet error rates.

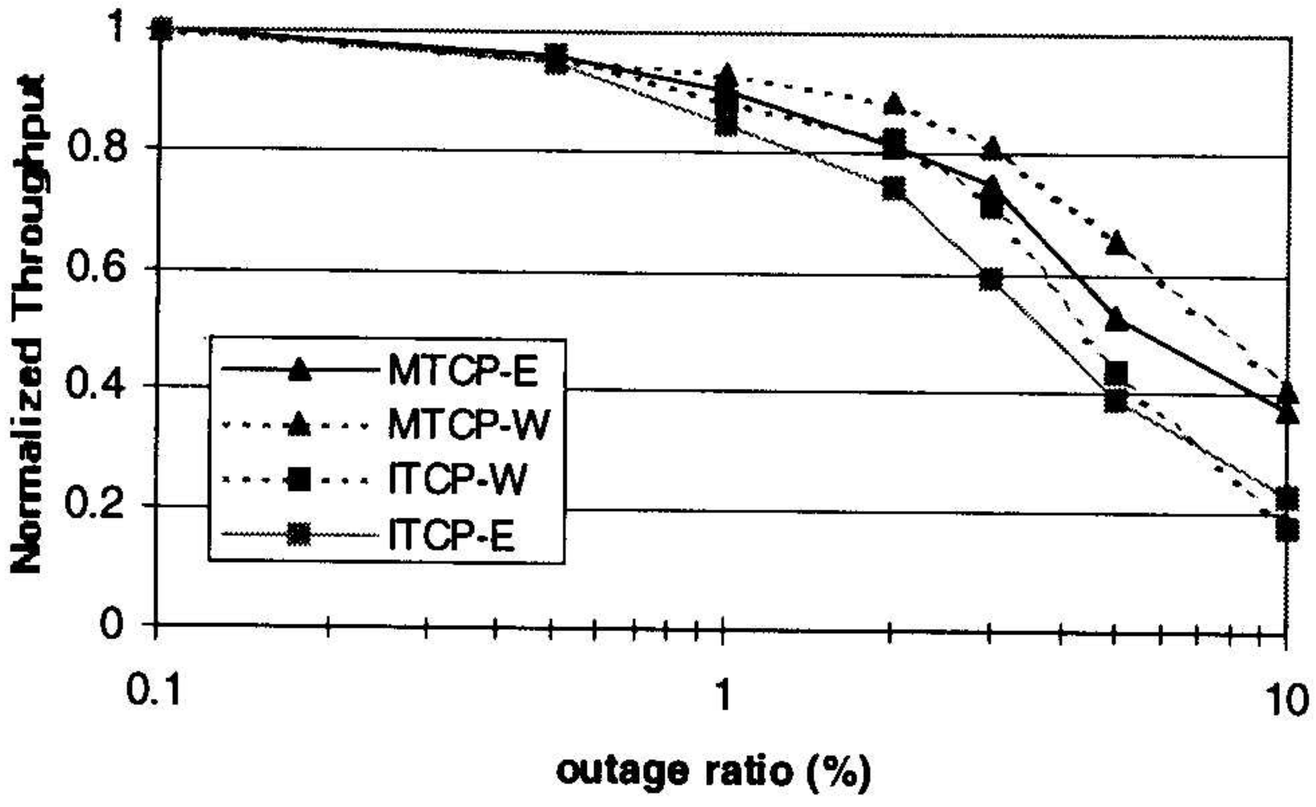

Figure 12b: Comparison of normalized throughput in wireless and ethernet configurations at different outage ratios.

compare the performance of the protocols at different outage ratios. The outage ratio, which is independent of the data rate, can be defined as the percentage of the time spent in the BAD state. The average length of an outage is kept constant at 10,000 Wireless-bit times, equivalent to the length of 10,000 Wireless bits in time. Since the ethernet data rate is 11 times faster than the Wireless data rate, the outage time corresponds to 110,000 Ethernet bits. The results of our experiments, shown in Figure 12b, indicate that the performance of the protocols in the Ethernet and the Wireless cases is closer than in Figure 12a. This can be attributed to the fact that the packet error rate decreases as the data rate increases.

On an average, an outage causes a loss of 10,000 Wireless bits and 110,000 Ethernet bits, which roughly corresponds to 2 packets and 11 packets in Wireless and Ethernet cases, respectively (assuming the packets size is 12,000 bits). Note that 2 packets are much more likely to be lost in the Wireless case than 1 packet because most outages will tend to overlap into the successive packet. Consequently, the ratio of the number of packets lost to the number of bits lost in the Wireless case is 2:10000, while it is only 1:10000 in the Ethernet case. Thus, the packet error rate in the Wireless case is approximately twice as much as in the Ethernet case at a particular outage ratio, which explains the results shown in Figure 12b.

We also studied the impact of different packet loss rates on the real-time constraints of an application. Due to the difficulty in computing the end-to-end delay accurately, as clocks on two machines need to be precisely synchronized, we introduce a metric known as inter-buffer arrival time. The inter-buffer arrival time can be defined as the time elapsed between the reception of two successive socket buffers at the application layer. Since our network topology consists of fixed connection routes, we can assume that the end-to-end delay without the presence of errors is fairly constant. It can be deduced that the additional end-to-end delay experienced in presence of errors is manifested in the extra inter-buffer arrival time seen at the application layer. Thus, inter-buffer arrival time can provide an insight into the changes in the end-to-end delay as well as the waiting time seen by the application in receiving data. In our simulation, the socket buffer size was chosen to be 1460 bytes. This particular choice of socket buffer size is important, since it is equal to the maximum data that can be sent in a single TCP packet. We can infer that the inter-buffer arrival time roughly corresponds to the inter-packet arrival time of packets arriving in order, since each buffer is placed in a separate packet. Figures 13a and 13b show the mean and the standard deviation of the inter-buffer arrival time measured over the lifetime of the connection at various packet error rates. We can see that MTCP performs better than I-TCP and end-to-end TCP in keeping a relatively low inter-buffer arrival time at higher packet loss rates. Interestingly, the standard

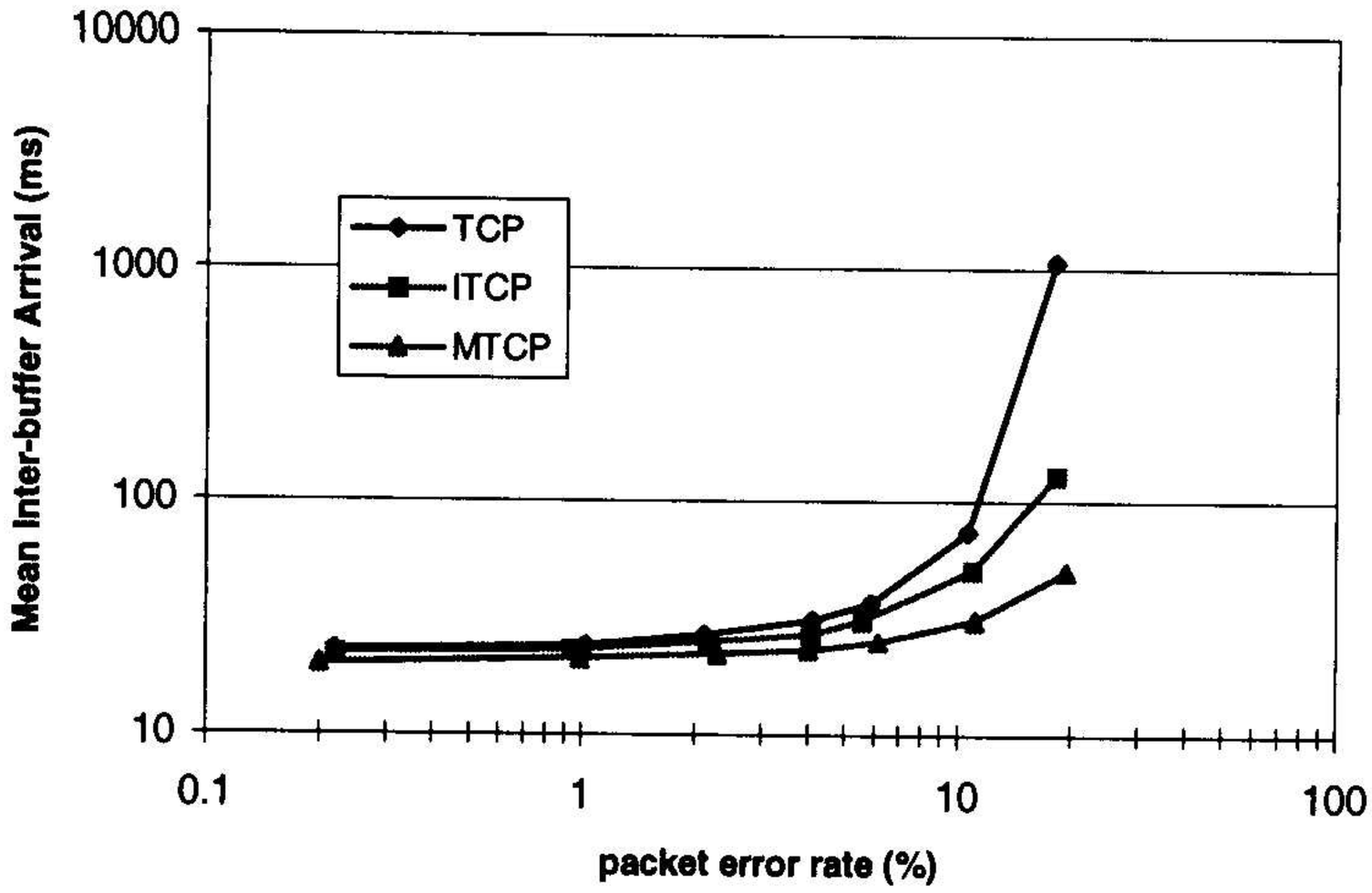

Figure 13a: Mean Inter-buffer Arrival Time vs. Packet Error Rate (Wireless Configuration)

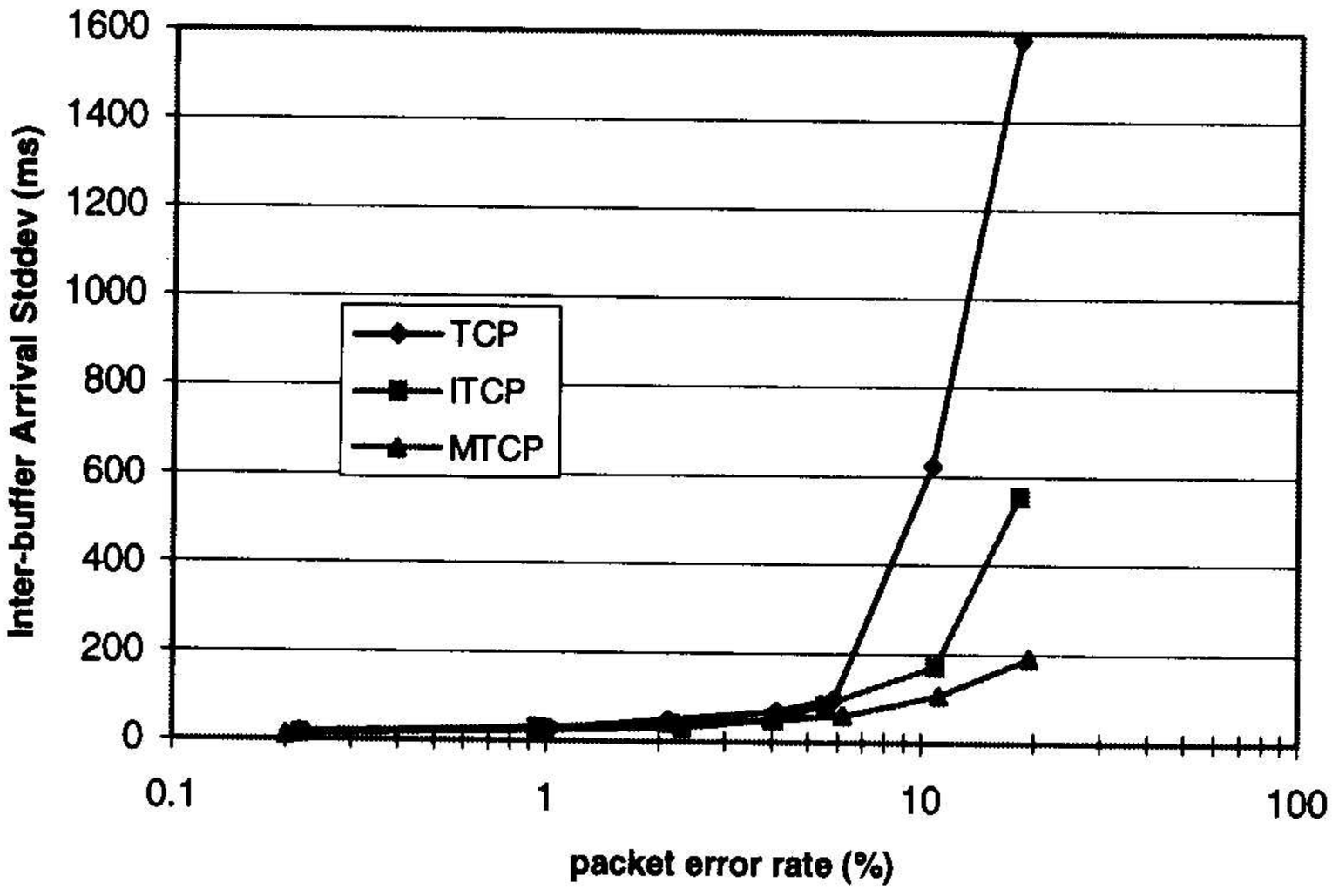

Figure 13b: Standard deviation of Inter-buffer Arrival Time vs. Packet Error Rate (Wireless Configuration)

deviation of the inter-buffer arrival time is also much lower in MTCP at higher packet loss rates, indicating that applications would be subject to less jitter and lower waiting time in receiving data. This is especially important for real-time applications, which are constrained by certain end-to-end delay and delay jitter requirements.

Protocol	Maximum throughput for bidirectional transfer (Kbytes/sec)	% CPU Usage
I-TCP	771.08	21
MTCP	866.06	12

Table 3: Throughput & CPU Usage for Bidirectional transfer under Ideal Conditions (for the ethernet configuration).

We now present various experimental results, which suggest an improvement in the utilization of the CPU resources by MTCP. As shown in Table 3, our experiments reported 12.3% difference in maximum throughput achieved by MTCP over I-TCP in the bi-directional data transfer tests for the Ethernet configuration. We claim that the bi-directional test is a more reliable indicator of processing limitations of a protocol, since it is not subject to various optimizations, as in the case of bulk-transfers. Specifically, factors such as the implementation of the Nagle's algorithm [16] and the timing of delayed acknowledgments can cause a significant impact on the throughput of unidirectional transfers. Higher throughput of MTCP under ideal conditions of the bi-directional tests suggests an improvement in the efficiency of protocol processing. Furthermore, the lower CPU usage by MTCP, as shown in Table 3, supports the implication that our protocol, in fact, reduces processing load on the end systems. We believe that majority of the contribution to this effect is due to the elimination of the IP layer. Removal of the IP layer from the TCP/IP stack provides benefits such as reducing protocol processing load, simplifying buffer management, decreasing the protocol header size, and possibly, reducing the number of memory copies. We also evaluated the impact of various packet error rates on the usage of CPU resources by each protocol according to the following metrics: Mean CPU Usage and CPU-Duration. Mean CPU Usage is simply the average % of the CPU used by the simple test application and the protocol during the lifetime of the connection. We subtract the background CPU usage (usage when no applications are running) from every CPU usage measurement in order to compute a more accurate estimate of the percentage of the CPU utilized in actually transferring the data. CPU-Duration approximates the area under the CPU usage curve as it varies over the duration of the connection. This measure is

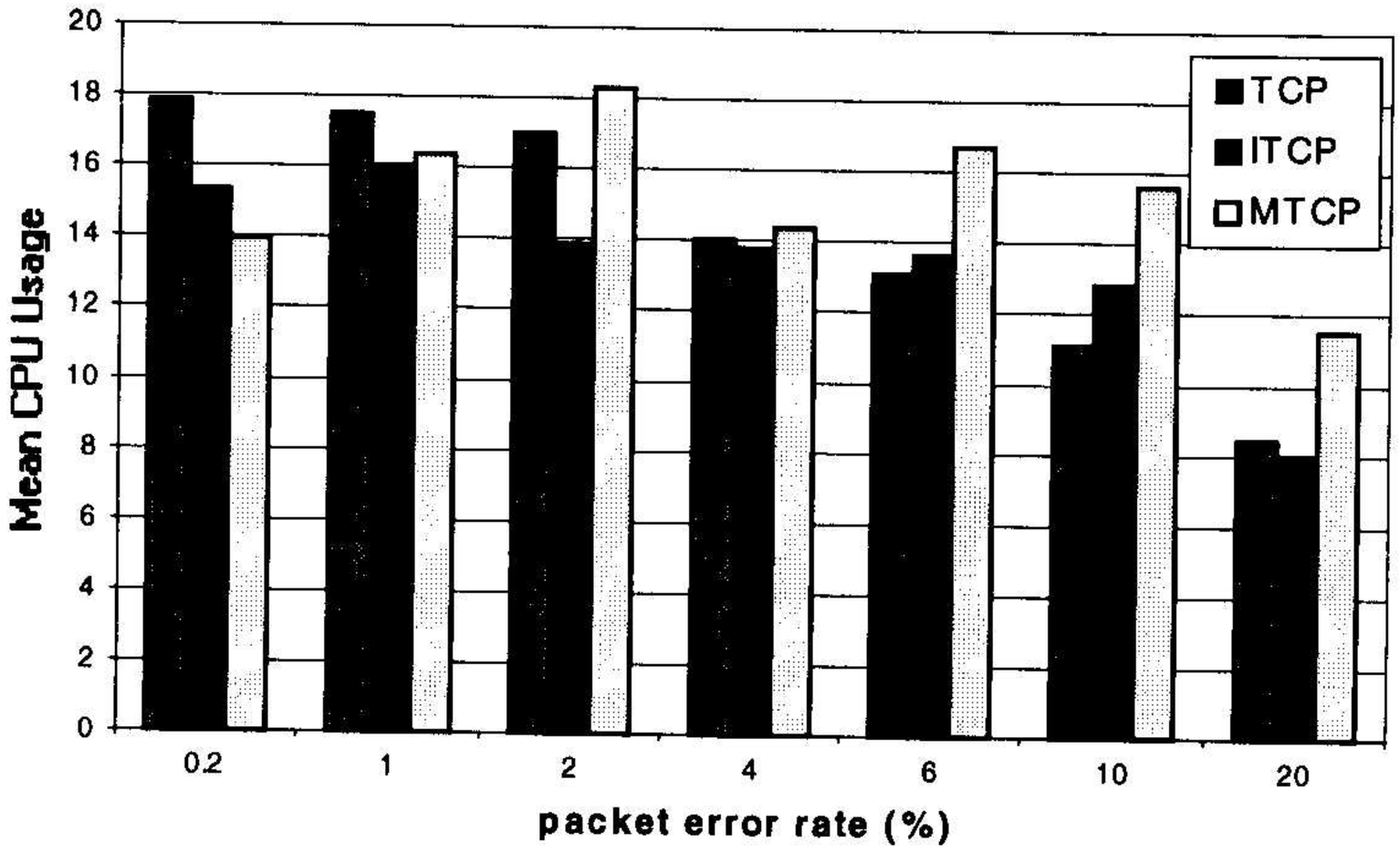

Figure 14a: Mean CPU Usage vs. Packet Error Rate (Wireless Configuration)

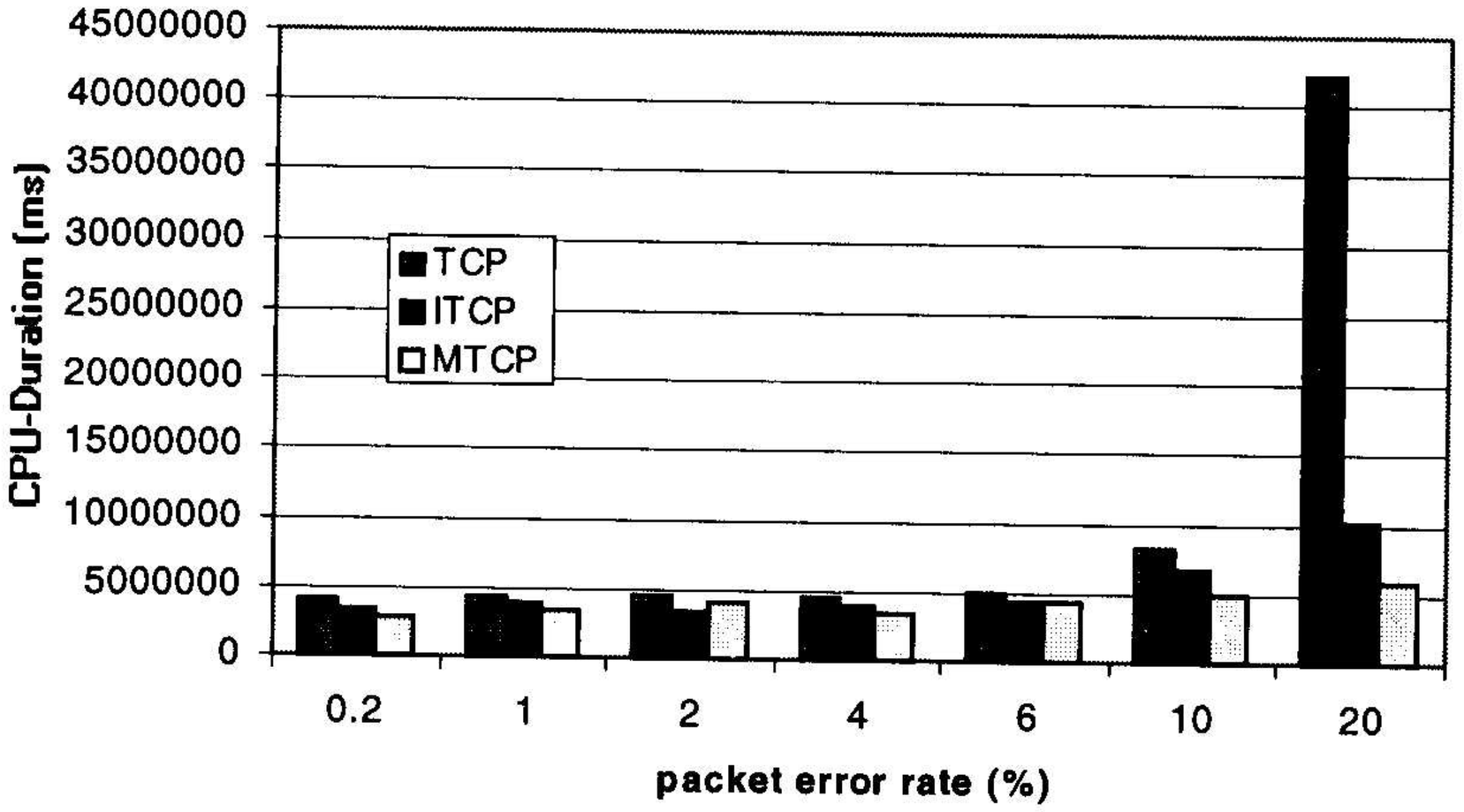

Figure 14b: CPU-Duration vs. Packet Error Rate for 1460*10000 byte transfer (Wireless Configuration)

estimated by taking the product of the average CPU usage and the time duration of the connection. The CPU-Duration signifies the total energy consumed by the application in order to transfer a certain number of bytes. The results of our experiments are shown in Figure 14a and 14b. In Figure 14a, we can observe that the CPU usage tends to reduce as the packet error rates increases and also that MTCP tends to maintain a higher utilization of the CPU than other protocols at higher packet error rates. This is explained due to the fact that there are more coarse timeouts in the case of I-TCP and end-to-end TCP, resulting in a less

frequent use of the CPU. On the other hand, since MTCP times out less often, it completes the data transfer quicker than the other two protocols, thus offsetting the higher CPU usage. This is illustrated in Figure 14b, which shows that MTCP consumes less energy than I-TCP and TCP in order to transfer data over the network. These differences are more pronounced at higher packet error rates where timeouts are more frequent in I-TCP and TCP than in MTCP and unnecessary energy expended on tasks such as handling protocol timers is greater, while no data transfer takes place.

6. Other Issues

6.1 Application Re-linking

One of the criticisms of the split connection approach, and specifically Indirect TCP, is that applications running on the mobile host have to be re-linked with the I-TCP library in order to use special socket calls for the setup of the indirect connection [11]. Our approach in solving this issue is to subsume the mechanisms used to create an indirect connection into the MTCP protocol, making the connection establishment process completely transparent to the standard socket calls. Furthermore, the MTCP protocol driver can be designed to replace TCP drivers in the kernel mode, while interfacing with standard socket calls of the network operating system. These techniques can be used to seamlessly integrate any application into the rest of the Internet.

6.2 Handoff Support

In a typical wireless network, mobile hosts connect to the fixed network through mobile-gateways, also known as Point of Attachments (POAs), which act as routers between the wireless subnet and the rest of the network. As a mobile host roams, it may leave the coverage area of the current POA. To maintain connectivity, a handoff process is invoked such that the mobile host reconnects itself to a new POA. Typically, the handoff process causes the

network layer to re-route all the packets to the new mobile-gateway via the old mobile-gateway [17].

The split-connection approach performs handoffs by migrating the state of all the indirect connections to the new mobile-gateway, while ensuring that the entire process is transparent to the connection endpoints on the mobile host and the fixed host. The details of the techniques involved in transferring the state are addressed in I-TCP [6]. We follow a similar strategy here, with the exception that a new *Connection ID* has to be reestablished for each socket connection on the mobile host. Specifically, as the *Connection ID* used on the old mobile-gateway might be in use on the new mobile-gateway, a new control packet is sent to the mobile host with the new value of the mobile-gateway's *Connection ID* on each connection [9].

The most commonly sited problem for Indirect-TCP is that it takes a longer time to complete the handoff process [11]. The main reason for this handoff latency is due to the complexity involved in migrating the state of all indirect connections associated with the mobile host. The state of a connection consists of the state of all TCP variables and also the data present in the TCP send and receive buffers at the time of the handoff. In order to reduce the handoff latency, we propose a scheme for setting up indirect connection, which optimizes the common case. Our scheme can be applied to Wireless Local Area Network (WLAN) technologies, such as those provided by Proxim or Wavelan. Figure 15 shows a typical WLAN configuration. In such configurations, mobile hosts connect to the wired network via devices known as Access Points (APs). An Access Point serves as a bridge between the Wireless Local Area Network and the Wired Local Area Network. Mobile hosts within the coverage area of the "nearest" AP can communicate with hosts on the wired network through the AP, which essentially acts as a basestation. When a mobile host moves into the coverage area of another AP with a stronger signal, it performs a handoff by registering itself with the new AP. Consequently, the new AP forwards all packets destined to the physical address (MAC address) of the handed off mobile host [15]. The main difference between a basestation and an Access Point is that

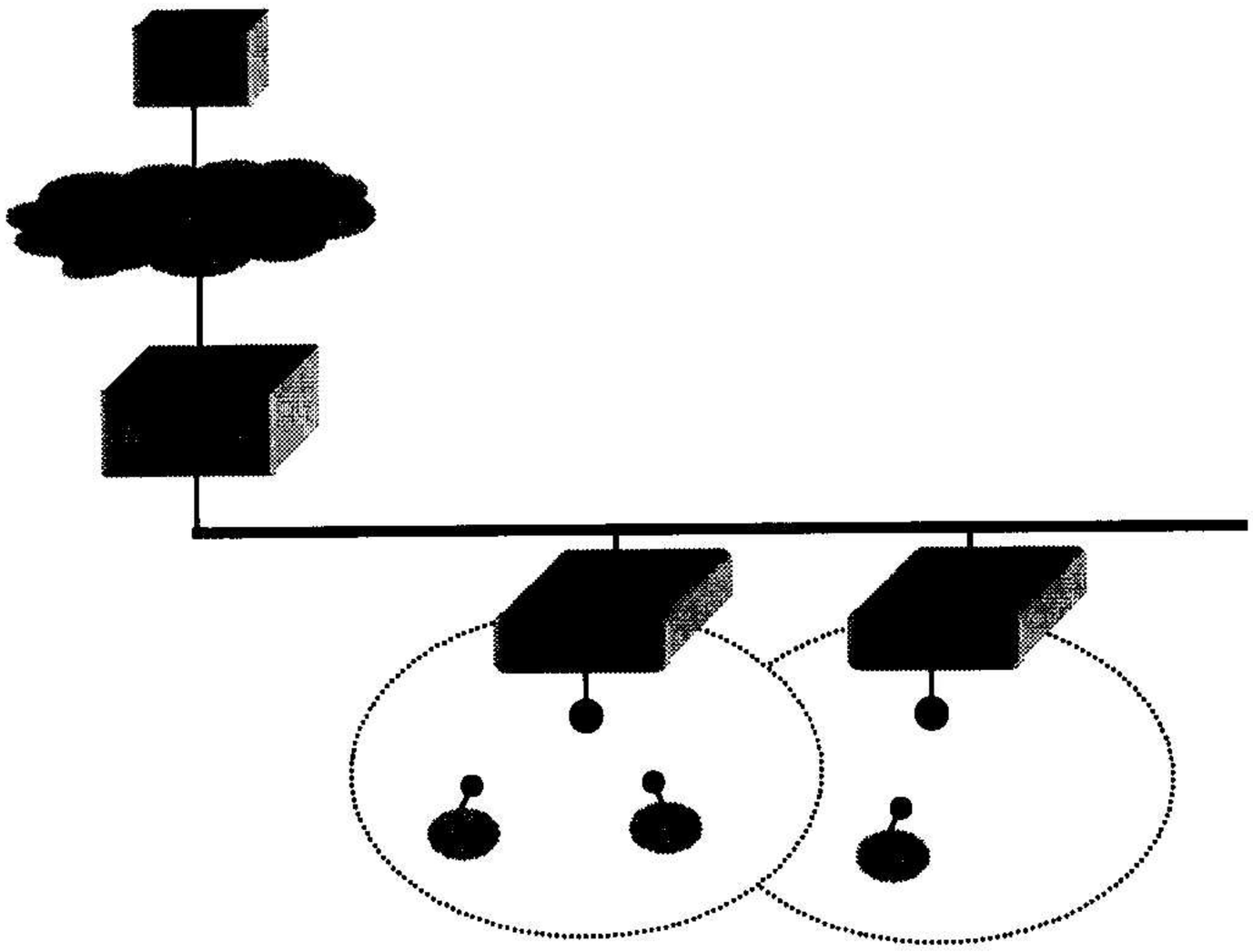

Figure 15: A Typical Wireless Local Area Network (WLAN)

the basestation is a router, while the Access Point is a bridge. Our scheme in the context of the present WLAN technologies is to split the end-to-end connection at the gateway and communicate with the mobile host over the AP using MTCP, as shown in Figure 16. This is possible since many WLAN technologies use Ethernet IEEE802.3 packet formats. The benefit of this scheme is that no transfer of state is required when mobile devices handoff within the same local area network, as handoffs are done on the MAC layer. In addition, we also get the benefit of using a streamlined protocol, such as MTCP, over the segment between the gateway and the mobile host. MTCP is particularly suitable for this segment, since the segment does not require any routing facilities in the protocol and, therefore, can be considered as a single hop link from the perspective of the transport layer. The packets that are lost during the handoff process can quickly be recovered, since these losses are handled separately by MTCP on the last segment. Essentially, losses occurring due to handoffs can be treated identical to the losses occurring on the wireless segment due to effects as interference and fading.

We propose the following technique for setting up the split connection in the WLAN configurations. Every mobile host registers itself with the MTCP-

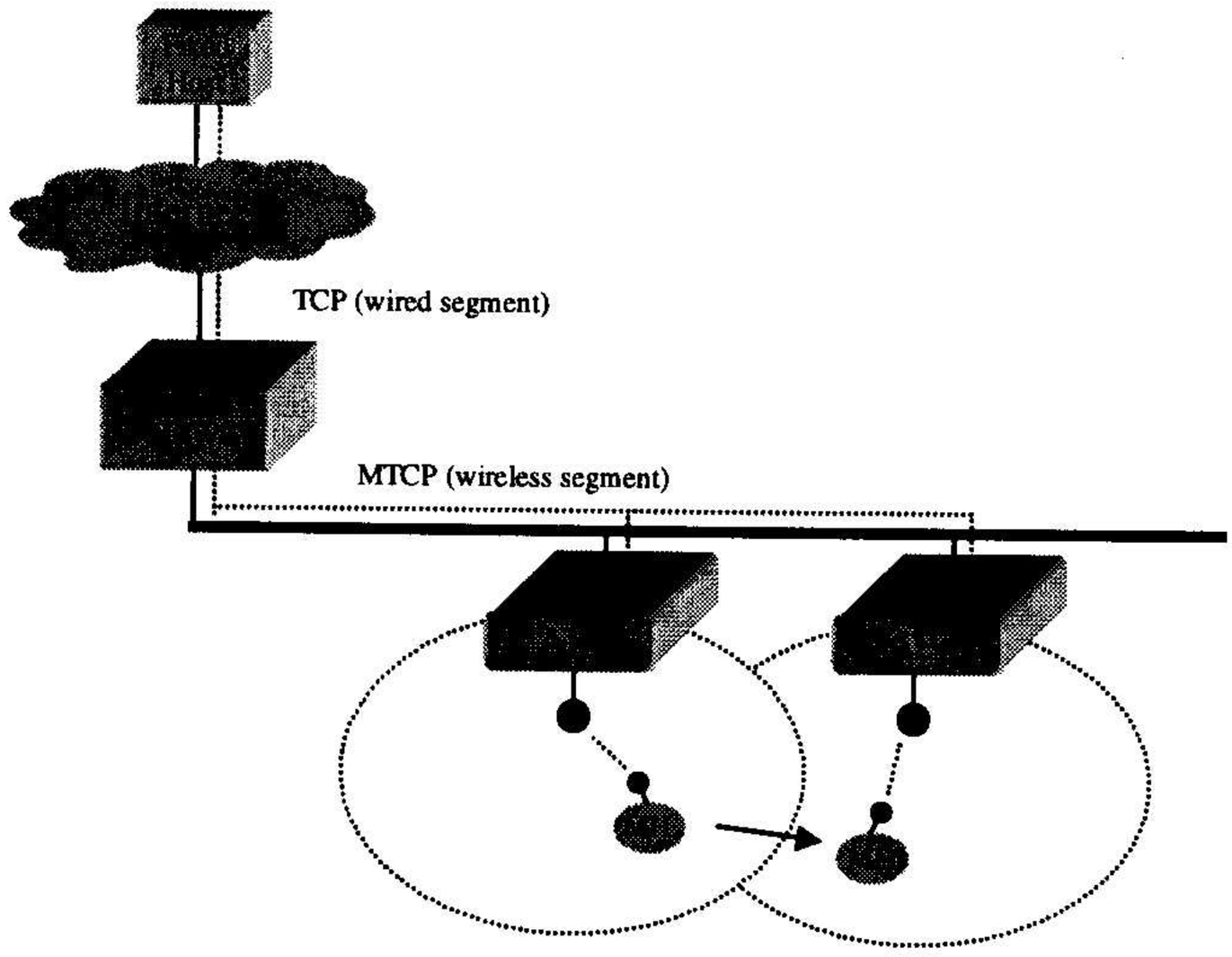

Figure 16: Split Connection using MTCP in WLAN

enabled gateway, which maintains a list of mobile devices in its local area network. The gateway acts like a transport-layer proxy, performing the function of setting up indirect MTCP connections on behalf of the registered mobile hosts and redirecting data from the wired segment to the "wireless" segment (the hop between the gateway and the mobile host) or vice versa. When a mobile host hands-off to an AP on another local area network, it un-registers itself from the previous gateway and registers itself with the new gateway. All packets destined to the mobile host are re-routed through the new gateway using Mobile-IP and the split connection-state is transferred from the old gateway to the new gateway much as in I-TCP. Note that the states of the split connections associated with a mobile still need to be transferred for an inter-local-area handoff. However, we argue that our scheme optimizes the common case of handoffs occurring within a LAN by eliminating much of the complexity.

7. Future Work

We are currently incorporating asymmetric protocol design techniques into MTCP to further offload processing from the mobile device. In an asymmetrically designed protocol, peer functions are implemented through

algorithms and procedures that are of substantially different complexity, with the lower complexity used on the mobile device [9]. We are also working on improving the efficiency and accuracy of our timer implementation.

In this paper, a simple burst error model was used to study the behavior of TCP and MTCP under specific situations. We are looking into other error-models, which characterize a wider range of wireless link operational conditions.

One of the important advantages of the split approach is that it enables the development of lightweight transport protocols, such as MTCP. However, the advantages come at the cost of maintaining significant amount of state at the basestation, making handoffs relatively slow [11]. In this paper, we proposed one scheme for optimizing the split connection model in a Wireless Local Area Network configuration. We are currently investigating techniques in which the split-connection approach, and specifically MTCP, can be designed to reduce latency involving transfer of state from one basestation to another for the general case.

Finally, it is an interesting research area to investigate how TCP can be efficiently adapted into networks with multiple wireless hops, such as Mobile Ad-hoc Networks, or into hybrid networks consisting of a mixture of wireless and wired links.

8. Summary and Concluding Remarks

In this paper, we have presented the design of a lightweight transport protocol (MTCP) that improves the end-to-end performance of TCP in networks with wireless links and also reduces processing requirements on mobile devices. The protocol is based on the split-connection model, where the connection between the mobile host and the fixed host is partitioned into the wireless segment and the wired segment. The key idea behind MTCP is in the recognition that the wireless segment is, in fact, a single-hop connection. This allows elimination of some transport-layer functions and simplification of other

functions, leading to a reduced communication load on the wireless segment and the mobile device.

The design of MTCP also incorporates various methods to efficiently recover from wireless losses. These techniques are simple to implement and our performance results indicate that they are very effective in sustaining good performance even under heavy packet losses. Our experiments have also highlighted the benefits of removing congestion control mechanisms from the wireless transport protocol. The elimination of such mechanisms enhances the effectiveness of the selective acknowledgment methods employed by MTCP. We have studied the behavior of MTCP and TCP at different data rates and different packet losses and our results indicate substantial improvements of MTCP over TCP in terms of the processing requirements and the overall throughput. It is important to point out that many of the techniques employed in MTCP, such as the elimination of the IP layer, enhancements in the error recovery mechanisms and handoff optimizations, can be applied to other approaches as well.

Finally, we have shown that MTCP can be designed to hide the process of establishing indirect connection from standard socket calls, such that applications do not need to be relinked. This addressed our goal of seamlessly integrating mobile applications into the Internet.

References

[1] G.H Forman and J. Zahorjan, *The Challenges of Mobile Computing*, IEEE Computer, April 1994.

[2] T. Imielinski and B. R. Badrinath, *Mobile Wireless Computing: Solutions and Challenges in Data Management*, Rutgers University, Department of Computer Science, Technical Report

[3] J.B. Postel. *Transmission Control Protocol*. RFC, Information Sciences Institute, Marina del Rey, CA, September 1981. RFC-793.

[4] W. R. Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley, Reading, MA, Nov 1994.

[5] R. Caceres and L. Iftode. *Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments*. IEEE JSAC, 13(5), June 1994

[6] A. Bakre and B. R. Badrinath, *I-TCP: Indirect TCP for mobile hosts*, in Proc. 15th International Conference on Distributed Computing Systems, Vancouver, Canada, June 1995, pp. 136-143.

[7] H. Balakrishnan, S. Seshan, and R. H. Katz. *Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks*, Wireless Networks, Vol. 1, No. 4, pp. 469-481, Dec 1995.

[8] R. Yavatkar and N. Bhagwat. *Improving End-to-End Performance of TCP over Mobile Internetworks*. In Mobile 94 Workshop on Mobile Computing Systems and Applications, December 1994.

[9] Z.J. Haas, *Mobile-TCP: An Asymmetric Transport Protocol Design for Mobile Systems*, ICC'97, Montreal, Canada, Jun 8-12, 1997.

[10] B. R. Badrinath, A. Bakre, T. Imielinski, and R. Marantz, *Handling Mobile Clients: A case for indirect interaction*, in Proc. 4th Workshop on Workstation Operating Systems, October 1993.

[11] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. H. Katz., *A Comparison of Mechanisms for Improving TCP Performance over Wireless Links*, In Proceedings of ACM SIGCOMM 96.

[12] S. Keshav and S. Morgan. *SMART Retransmission: Performance with Overload and Random Losses*. In Proc. Infocom '97, 1997.

[13] V. Jacobson, *Compressing TCP/IP Headers for Low-Speed Serial Links*, RFC 1144.

[14] Microsoft Developer Network (MSDN) Library, Jan 1998.

[15] Proxim RANGELAN2 7100: User's Guide, 1995.

[16] Nagle, J. 1984. *Congestion Control in IP/TCP Internetworks*, RFC 896, 9 pages (Jan.)

[17] C. Perkins, editor, *IP Mobility Support*, Internet Engineering Task Force, Internet Draft, draft-ietf-mobileip-protocol-15.txt, 9 Feb. 1996.