Journal of High Speed Networks 10 (2001) 187–207
IOS Press

# The design and performance of Mobile TCP for wireless networks

Zygmunt J. Haas and Abhijit Warkhedi
*School of Electrical Engineering, Cornell University, Ithaca, NY 14853-2801, USA*
*E-mail: haas@ece.cornell.edu, warkhedi@cisco.com*

**Abstract.** In this paper, we propose a novel approach to reduce the wireless communication overhead associated with the transport layer. Portability is a key element of mobile computing. As mobile devices shrink in terms of size and weight, their computing capabilities are also reduced, constrained by battery capacity. To sustain a good performance level of mobile applications, we devise a solution that reduces the processing complexity on computing-limited mobile devices. Additionally, we aim to minimize the use of wireless resources to further improve performance. Specifically, we develop a streamlined protocol architecture, Mobile TCP (MTCP), that achieves the elimination of IP processing on the wireless segment of the TCP connection. MTCP operates over a single hop wireless link, and, consequently, it eliminates the unnecessary overhead placed on mobile devices, such as the TCP congestion control mechanisms. In addition, we investigate the impact of streamlining the socket layer on offloading the processing overhead. Our experimental results indicate a substantial improvement in the efficiency of protocol processing. For instance, results show that the MTCP processing time per packet is approximately only one fourth that of TCP and the use of CPU resources is reduced by up to 50%. Furthermore, the protocol incorporates various robust, yet simple, loss recovery techniques to considerably improve the throughput in lossy wireless conditions.

## 1. Introduction

The vision behind mobile computing [1,2] is to support ubiquitous access to various forms of information, such as data, voice and 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 requirements of the portable mobile devices and the special characteristics of the wireless link.

Portability is an important element of mobile computing. As mobile devices shrink in physical parameters (size and weight), their computing and power capabilities are also reduced, constrained by battery capacity. Unfortunately, battery technology has been improving only at a modest pace in terms of increased capacity per unit weight and volume [3]. To maintain a constant level of performance of mobile applications, there is a need to minimize processing load on portable mobile devices [4].

The primary focus of our work is the design of transport protocols in computing-limited wireless environments. We devise a lightweight protocol architecture, Mobile TCP (MTCP), that allows the emulation of TCP functionality between a mobile host and a stationary host, in a way that reduces the processing load on the mobile host and minimizes the use of the wireless medium. Most of our protocol enhancements stem from the fact that the communication between the mobile host and the basestation is over a single link. Thus, the design of MTCP resembles that of a link-layer protocol. Consequently, as we see in this paper, many functions can be either simplified or totally eliminated in the wireless segment of the TCP connection, leading to a lean protocol code on the mobile machine. Furthermore, our implementation streamlines the protocol stack to additionally offload processing from the mobile devices.

Our protocol implementation is based on the split-connection model, where the connection between the mobile host and the basestation is split into two connections: the connection between the fixed host and the basestation, and the connection between the basestation and the mobile host. The first connection is called the wired segment, while the second portion is the wireless segment. The division, by itself, is not a new idea and was already employed in previous studies [5,6]. We chose to employ the split-connection approach because it provides a convenient proxy-style model to enable the development of a new lightweight protocol on the wireless segment. It should be noted, however, that MTCP could be implemented using other approaches as well. We address this point in Section 6.

In addition to reducing the communication overhead, MTCP incorporates various robust yet simple techniques to combat heavy losses over the wireless link. Losses are prevalent on networks with wireless links due to various factors, such as noise, interference, channel fading, and user mobility. It has been identified that TCP performs poorly in wireless networks, since it assumes all losses occur as a result of congestion problems in the network. In response to a loss, TCP [7,8] takes various steps to throttle its transmission rate to alleviate congestion. However, this leads to a degraded performance in networks that exhibit non-congestive losses [9]. Several studies 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 [6,10–12]. The split-connection model, which is the basis of our implementation, addresses this issue by separating loss recovery on the wireless segment from the wired segment. Thus, in this paper, our aim is *not* to solve the congestion issue, but rather to improve loss recovery on the wireless segment.

In accordance with the goals of our research, we evaluate the performance of MTCP in terms of its ability to offload processing from mobile devices, as well as its ability to improve throughput in lossy conditions. Our measurements, which are based on metrics such as throughput, protocol latency, CPU-usage, CPU energy consumption, and inter-buffer delay, indicate that our protocol is indeed effective in addressing both of these goals.

The rest of this paper is organized as follows. Section 2 briefly summarizes some previously conducted work on energy conserving protocols for portable devices. This section also outlines some solutions that have been proposed to address the performance issues of TCP over networks with wireless links. In Section 3, we present the design of MTCP followed by some implementation details. Section 4 describes our experimental testbed, including the error model and performance metrics for evaluating the protocols. Section 5 presents the performance results and analysis of several experiments. In Section 6, we propose some alternative approaches that can be used to implement MTCP. We discuss our future plans in Section 7 and conclude with a summary in Section 8.

## 2. Related work

Considerable research work in the past has focused on improving the performance of TCP over wireless and other lossy links, as it is the most dominant protocol in the Internet today. Some work has also been undertaken in developing energy efficient protocols and techniques for portable devices. In this section, we present a brief overview of each area of research.

### *2.1. Energy efficient protocols and techniques*

Many portable devices today are equipped with power-saving components. Two such components are the processor and the wireless communication device [3]. Processors typically offer two types of power saving features. One power-saving feature is the ability to slow down the clock rate. Energy consumption ($E$) is essentially proportional to the switching capacitance ($C$), and the square of the voltage ($V$) ($E \propto CV^2$). Consequently, energy consumption can be reduced with a decrease in clock rate or by a reduction in voltage. Various software techniques can be employed to dynamically change the speed of the clock [13,14]. The general strategy is to dynamically adjust the CPU clock depending on the processor utilization during a particular time interval. Another power-saving feature of the processor is the ability to shutdown the processor when the system is idle and turn it back on when the next interrupt occurs. A well-designed operating system can deduce whether the system is idle using the current state of all processes. Whenever all the processes are blocked, the processor is turned off and subsequently restarted upon the occurrence of an interrupt [15–17]. Examples of operating systems that use this strategy are Windows [18,19] and UNIX. Wireless communication devices also provide similar power-saving features that enable the devices to enter low-power consumption mode during periods of inactivity.

From protocol design standpoint, general strategies to promote the transition of processors and communications devices into energy conserving modes can be stated as the following: minimize the processing complexity of protocols, and decrease the overall use of the network device [3,20] without sacrificing any functionality. Various power-saving techniques have been proposed at different layers of the protocol stack.

A link-layer error control scheme, proposed by [21], conserves energy by reducing the data transmission rate when the channel is impaired, so that the number of unnecessary transmissions is minimized. While the channel is presumed to be impaired, short probe packets are continuously sent until some feedback is received for these packets. The protocol proceeds with the transmission of data packets upon the receipt of the first feedback packet indicating an improvement in the link quality. This scheme has been shown to be more energy efficient, however, with a slight loss in throughput.

Another link-layer scheme, proposed by Lettieri et al. [22], follows an adaptive error control strategy that uses Forward Error Correction (FEC) and Automatic Repeat reQuest (ARQ) to minimize energy consumption. ARQ mechanisms, such as Go-Back-N (GBN) and Selective Repeat (SRP), achieve reliability by retransmitting lost packets. Although ARQ mechanisms are relatively simple, they can cause a significant number of retransmissions when losses are excessive. FEC, on the other hand, reduces retransmissions by correcting erroneously received packets, however, at the cost of greater computational load, latency, and overhead in the form of additional bits. The adaptive error control scheme investigates the tradeoff between ARQ and FEC, and concludes that optimum energy savings can be achieved through an adaptive adjustment of the two mechanisms.

AIRMAIL [23] is a reliable link-layer protocol, which uses asymmetric protocol design to reduce processing load on the mobile device, while placing much of the complexity on the basestation. This protocol also minimizes the amount of data transmitted over the wireless channel by combining several acknowledgments into a single one, thus conserving power at the cost of increased latency. In addition, AIRMAIL employs FEC to minimize the energy expended on retransmissions in highly lossy conditions.

Another approach to minimizing the utilization of the communication device is to use header compression. In [24], the authors propose a low-loss header compression technique for TCP/IP, which significantly improves throughput performance of TCP. Their study also indicates that Van Jacobson header compression algorithm [25] can cause throughput degradation in lossy conditions due to inconsistencies in the compression-state.

Another energy conserving strategy is to use a medium access protocol that dictates in advance when each wireless device may receive data. This allows each device to sleep when no data is to be received. Variations of this technique are employed in 802.11 LAN standard [26], LPMAC [27], POCSAG protocol, and Motorola's FLEX™ protocol [28].

Considerable work has also been done in the area of lightweight transport protocols. This work is motivated by the need to provide a low overhead protocol, without performance bottleneck for high-speed networks. To minimize processing bottlenecks, lightweight protocols employ various techniques, such as smaller header sizes, simpler flow and congestion control strategies, and partial or complete implementation of the protocol in hardware. Similar ideas can be applied for reducing the energy consumption and the processing overhead on mobile devices. Examples of experimental lightweight protocols include VMTP [29], LNTP [30], high-speed transport protocol [31], Delta-t [32], and XTP [33].

### *2.2. Wireless TCP optimizations*

Figure 1 shows a typical network model for wireless networks. There are fundamentally three different approaches to improving TCP performance in the presence of non-congestive losses, such as those on wireless links [34]. The first approach is to use a reliable link-layer protocol on the wireless link in order to hide wireless losses from the TCP sender. Link-layer solutions recover from losses either by retransmitting lost packets (ARQ) or by correcting erroneously received packets (FEC). Although link-layer protocols fit well into the layered structure of network protocols, they can cause adverse interactions with the error recovery mechanisms of higher layer protocols, such as TCP. Specifically, competing retransmissions between the link-layer and the transport-layer caused by incompatible timer settings can result in redundant retransmissions [35]. Another concern is that error recovery mechanisms (such as ARQ, FEC, checksums) are duplicated at both the layers to provide reliability. This duplication of effort can impose a greater processing demand on mobile devices. TCP-aware link-layer solutions, such as Snoop [11], solve some these problems. Snoop introduces an agent at the basestation to cache TCP segments sent across the link. The agent detects losses by leveraging upon acknowledgments sent by TCP. This way it avoids additional control exchanges to perform error recovery. Upon detecting a packet loss, the Snoop agent retransmits the lost packet from the local cache and suppresses any duplicate acknowledgements heading toward the TCP sender in order to shield it from non-congestive losses.

The second approach is to split the end-to-end TCP connection into two connections: one instance of TCP operates over the wired segment and another instance of TCP or a different protocol operates over the wireless segment. A transport layer proxy (referred to here as the *Redirector*) at the basestation bridges the two segments together. An important consequence of the split-connection model is the separation of loss recovery on the wireless segment from the wired segment. As a result, losses over the wireless link can be recovered locally, while shielding the TCP sender on the fixed host from such losses. Indirect TCP (I-TCP) [5,10] is a split-connection solution that uses standard TCP over the wireless segment. The design of I-TCP is particularly intended for shielding TCP from packet losses during handoffs. I-TCP exploits the fact that lost packets are buffered at the basestation in the split-connection model. As part of the handing off process, I-TCP transparently moves the state of all connections (including send/receive buffers) associated the mobile from the old basestation to the new one. This is followed by an immediate retransmission and recovery of unacknowledged packets in the connection buffers, without the knowledge of the TCP sender on the fixed host. Studies, however, show that I-TCP is ineffective at recovering from wireless losses due to its choice of standard TCP for the connection over the wireless links [34]. Alternatively, a transport protocol with faster recovery mechanisms can be substituted, as in [6]. One problem with the split-connection model is that end-to-end semantics of TCP acknowledgments (ACKs) are not preserved. Specifically, ACKs for segments can now reach the source even before packets are delivered to the mobile host. However, proponents of this approach argue that most applications using TCP also have some kind of support for application-layer acknowledgments and error recovery [10]. In [12], the authors develop a protocol that does not violate the end-to-end semantics of split-connection approach by taking advantage of the persist state in TCP. Specifically, the source is completely throttled by shrinking the advertised window when the mobile host is disconnected, and activity is resumed as soon as connectivity is restored. Another technique called TCP splicing [36] preserves semantics by designing the transport proxy in such a way that ACKs (destined to the source) are not generated until the receiver acknowledges the forwarded data.

The third class of solutions, the end-to-end schemes, improve TCP performance by distinguishing between congestive and non-congestive losses [34]. Specifically, an agent residing on the basestation marks all duplicate ACKs corresponding to lost packets in order to explicitly notify the sender of non-congestion losses. The sender uses this information to decide whether to invoke congestion control mechanisms or not. One problem with the end-to-end schemes is that they are often slow to detect and recover from losses, since recovery is not performed locally. This results in considerable throughput degradation in WAN environment, where the round-trip delays are much greater than in a LAN [10,34]. Another major problem with the end-to-end approach is the need to modify the existing TCP implementations on the fixed host, making the solution impractical for wide scale deployment.

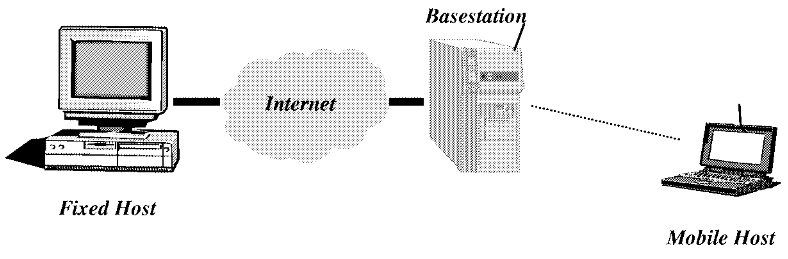

Fig. 1. Wireless network model.

## 3. Mobile TCP (MTCP)

### 3.1. Protocol description

Much of the previous research work deals with reducing processing and communication overhead at lower-layers of the protocol stack. The general strategies applied are reduction in code complexity, minimization of retransmissions, and header compression. Our approach is to focus on minimizing the protocol processing overhead at the upper layers of the protocol stack, specifically the transport and the network layer, where much of the protocol complexity exists.

In this paper, we propose a streamlined protocol architecture for TCP that reduces the communication load on mobile devices. Our approach employs the split-connection model, in which the end-to-end TCP connection is split into the TCP connection over the wired segment and the MTCP connection over the wireless segment. It should be noted that we use the split-connection model to design a new lightweight protocol (MTCP) on the mobile device, rather than to address the congestion issue. The split-connection model is not the central point of our work, and we provide some alternative implementation techniques in Section 3.6.

MTCP is based on our previous work in this area, as described in [4]. The key idea influencing our protocol design is the fact that MTCP operates over a single link, the wireless segment. Thus, in many aspects, the design of MTCP resembles that of a link-layer protocol. Furthermore, since the protocol executes over a single hop, all packets arrive in order and all losses on the wireless segment are non-congestion related. As a consequence, the design of MTCP can be greatly simplified and optimized, resulting in a significant reduction in processing and communication load over the wireless link. In addition, MTCP is tuned for quick recovery from losses on the wireless channel in order to improve the end-to-end TCP performance. The features of MTCP can be outlined as follows:

*Elimination of IP processing*: Because MTCP connection operates over a single link, there is no need for routing functionality provided by the IP layer. Addressing and multiplexing features of the TCP/IP stack are incorporated into MTCP. Note that we do not eliminate the IP layer, since it is still required for UDP and other IP-based protocols. However, IP processing is eliminated from protocol processing path for mobile applications that rely on TCP for reliable data transport. The MTCP protocol architecture is shown in Fig. 2.

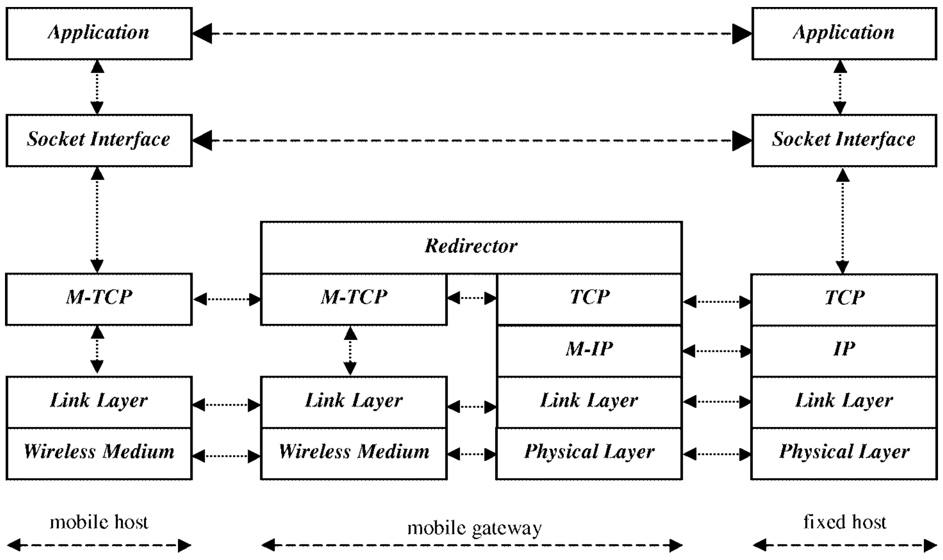

Fig. 2. Mobile-TCP protocol stack.

*Eliminate all congestion control mechanisms*: All losses over the wireless link are non-congestion related. This allows us to design MTCP without congestion control mechanisms. As a result, the flow control problem simply transforms into the problem of matching the rate of data flow with the receiver in order to prevent buffer overflow. A sliding-window flow control protocol is adopted to provide a simple and an efficient solution.

*Loss-less header compression*: We reduce the size of packet header primarily by eliminating the IP header and by compressing the source and the destination IP addresses (including port numbers) into unique connection identifiers.

*Optimized loss recovery techniques*: All packets on the wireless segment arrive in order. Our error recovery techniques are designed to be simple and effective in recovering from multiple packet losses. These techniques include selective acknowledgments (SACKs) designed for burst losses, explicit retransmission requests, and fast recovery from retransmission failures.

We now proceed with the detailed description of MTCP design. The next few sections deal with the connection control, flow control, and error control mechanisms of the protocol. In Section 3.7, we present some key implementation details of MTCP, focusing on the reduction of protocol complexity.

## 3.2. Connection control

MTCP uses a three-way handshake protocol, similar to TCP, in order to perform setup and tear down of 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 *Redirector* on the basestation (BS) acts as an intermediary, coordinating connection setup between the mobile host (MH) and the fixed host (FH). When the MH executes an active open, it communicates to the BS 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 BS. The MH conveys this information using a connection control packet shown in Fig. 3a. The BS then proceeds with setting up the connection with the FH using the destination IP address and port number specified by the MH. Similarly, when a MH executes a passive open, it informs the BS that it is prepared to accept connections. This allows the BS to accept or reject connection requests from FHs without communicating a priori with the MH. Similar technique is used at the BS for connections initiated by fixed hosts.

A consequence of eliminating IP headers is that MTCP has to subsume IP addressing functionality into its own packet headers. Instead of sending the full source and destination addresses (including TCP port numbers), we use a simple loss-less compression technique to reduce the amount of wirelessly transmitted data. Since all packets from the MH travel through the BS, we fold the IP addresses and the port numbers into unique connection identifiers. Specifically, at the time of connection establishment, a connection ID (CID) is assigned in each direction and is

| 0 | | 16 | 32 |
|---|---|---|---|
| | Code Bits | Connection ID | |
| Sequence Number | | Acknowledgment Number | |
| Reverse CID | | Checksum | |
| Max Buffer Size | | Max Packet Size | |
| Destination Port | | Source Port | |
| Destination IP Address | | | |
| Source IP Address | | | |

Fig. 3a. Connection control packet.

| 0 | | 16 | 32 |
|---|---|---|---|
| | Code Bits | Connection ID | |
| Sequence Number | | Acknowledgment Number | |
| Window Size | | Negative Ack Number | |
| Data Length | | Checksum | |

Fig. 3b. Data packet.

used in subsequent exchanges of data over the wireless segment. ($\mathrm{CID}_{\mathrm{MH}}$ represents the CID from MH to BS and $\mathrm{CID}_{\mathrm{MG}}$ is the CID in the reverse direction). The result is a much smaller user data packet header, as shown in Fig. 3b. Note that the first bit is used to distinguish between the data and the connection packet formats. MTCP uses the three-way handshake to establish connections and exchange unique connection identifiers between the MH and the BS. The bindings between CIDs and IP addresses 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 $\mathrm{CID}_{\mathrm{MH}}$ at the MH. At the BS, the $\mathrm{CID}_{\mathrm{MH}}$ 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 forwarding 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 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 the 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 can be transparently embedded into MTCP. We address the issue of application re-linking on the mobile host in Section 3.6.

### 3.3. *Flow control*

The flow-control problem in MTCP is much simpler than the generic problem that needs to be addressed by TCP. Since there is no congestion on the wireless segment of the connection, flow-control in MTCP simplifies into the problem of matching the rate of data flow with the receiver rate to avoid buffer overflows. Thus, congestion-control mechanisms [7,8] that are applied to TCP flow-control can be entirely excluded from MTCP.

Our design strategy is to develop a simple yet an efficient flow-control scheme for MTCP. Overflow of the receive-buffer can adversely affect performance by unnecessarily causing coarse-timeouts, wasteful retransmissions, and greater use of the CPU resources. The effects are more pronounced in the mobile environment, which is limited in terms of power, CPU, and bandwidth. Flow control in MTCP is achieved by using the sliding-window technique. The flow-control scheme is identical to the one used by TCP, except that congestion control mechanisms are not implemented. As a consequence, the flow-control window size is entirely controlled by the receiving end and the maximum size of the window is limited by the size of the MTCP receiver buffer. An optimal buffer size should be as large as the bandwidth-delay product of the connection. 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 multi-hop networks, large windows 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 windows significantly improve the chances of detecting packet losses without resorting to coarse 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 in TCP. We reserve this issue of dynamically choosing the optimal buffer size for our future research.

We note that the sliding-window protocol has been employed in lightweight protocols, such as eXpress Transport Protocol (XTP) [33], where the design of protocol features is governed by the feasibility of VLSI implementation. Thus, sliding-window protocol can not only be considered as an effective scheme, but also one that is of sufficiently low complexity.

### 3.4. *Error control*

Error control in TCP is implemented at 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 missed sequence numbers. When the receiver detects an error, it signals the sender of the loss using a duplicate acknowledgment (ACK). In turn, the sender identifies losses either by the arrival of three duplicate ACKs or by the absence of an ACK within an

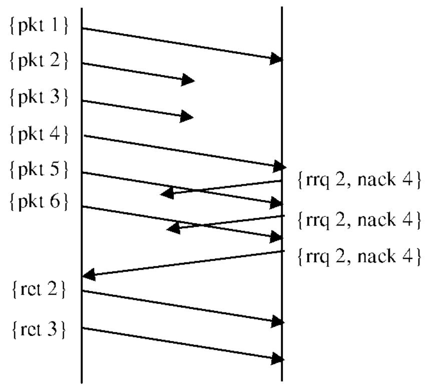

Fig. 4. MTCP SACK.

estimated timeout period. Upon identifying a packet loss, the TCP sender retransmits the lost packet and invokes various congestion control mechanisms, which throttle the transmission rate. In essence, TCP takes a conservative approach to loss recovery by assuming that only a single packet is lost. This technique is well suited for networks that experience light congestion, but too slow to recover in environments with heavy losses.

The error-control strategy in MTCP differs from that of TCP in a few key aspects that make it far more robust in recovering from highly lossy conditions. These differences can be stated as the following:

(1) *Explicit retransmission request (RRQ)* – All packets on the wireless segment arrive in order. Therefore, the first notification of loss in form of a duplicate ACK is sufficient for MTCP to trigger a retransmission, instead of waiting for three duplicates. We refer to this as explicit retransmission request, because each duplicate ACK unambiguously signals the loss of one or more packets.

(2) *Selective acknowledgments (SACKs) designed for burst losses* – Losses on wireless links typically occur in bursts. MTCP adopts a simple error recovery technique based on selective acknowledgments (SACKs) proposed in RFC 2018 [37]. In MTCP, every duplicate ACK contains the sequence of the next in-order segment (NIS), which conveys the size of the burst loss to the sender. Since this information is repeated in every duplicate ACK, the MTCP sender obtains an accurate knowledge of burst losses even if some ACKs are lost. Figure 4 illustrates this technique. In this example, the receiver generates three requests for the retransmission 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. Our performance results indicate that this scheme is quite effective in combating burst losses.

(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 them. 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 Fig. 5a. 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 Fig. 5a, the retransmission requests (RRQs – same as duplicate ACKs) 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 RRQs. One possible solution is to use SMART-style [38] sequence number of the received packet in the ACK 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 bit (RC) and every ACK in the opposite direction contains a retransmission-cycle reply bit (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

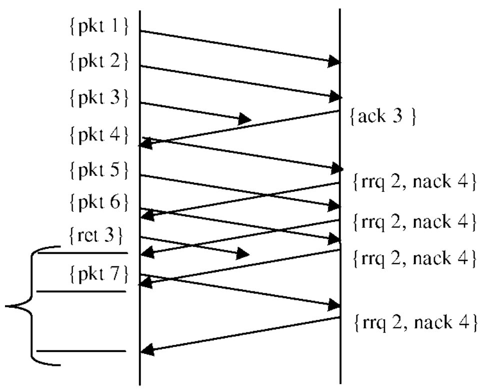

Fig. 5a. Problem.

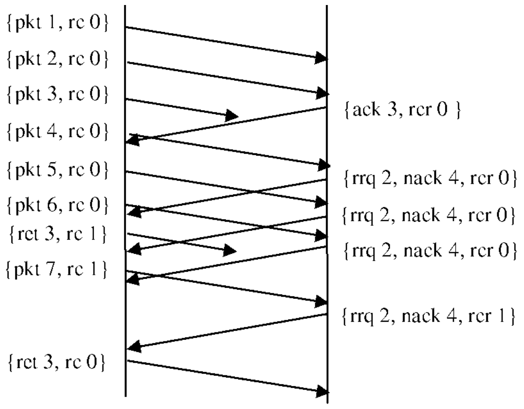

Fig. 5b. Solution.

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 prevents the sender from responding to every duplicate retransmission request. Figure 5b 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 for single or multiple retransmissions.

The combination of the above techniques makes MTCP robust and efficient in dealing with excessive packet losses that are bursty in nature. Moreover, the lack of congestion control further reduces the occurrence of timeouts, thus increasing the effectiveness of the error-control techniques.

## 3.5. *Handoff management*

In a typical wireless network, a mobile host connects to the fixed network through its current basestation, which acts as a router between the wireless subnet and the rest of the network. As a mobile host roams, it may leave the coverage area of the current basestation and enter the coverage area of a neighboring basestation. To maintain connectivity, a handoff is performed such that all the traffic to and from the mobile host is re-routed via the new basestation.

The split-connection approach performs handoffs by moving the state of all the indirect connections to the new basestation, while ensuring that the entire process is transparent to the fixed host. The state transfer entails transferring the state of all TCP variables and connection buffers associated with the mobile. The details of transferring state are addressed in I-TCP and we follow similar techniques, with one exception. In MTCP, a new Connection ID (CID) has to be reestablished for each connection on the mobile host. Specifically, since CIDs used on the previous gateway might be in use on the new gateway, a new control packet is sent to the mobile host with the updated CIDs. All subsequent packets from the mobile host to the gateway use the new CIDs. As in I-TCP, after the handoff is completed, all packets are re-routed to the new location of the mobile using Mobile IP [39].

A widely stated problem with the state transfer is that it is slow, and therefore, increases the handoff latency. A local handoff scheme proposed in [40] suggests otherwise. This scheme relies on transferring link-layer retransmission buffers of up to four packets from one basestation to another in order to recover lost packets. The handoff latency achieved in this scheme is less than 10 milliseconds. It is, however, likely that the latency is low due to the fact that the buffer transfer is performed within the LAN. Also, the transfers are relatively short, since buffer size is restricted to only four packets.

We propose a slight variation of I-TCP that can reduce the handoff latency. Instead of initiating the state transfer after the mobile has finished handing off, we propose to initiate the state transfer when the handoff is anticipated. This is based on the idea employed by multicast-based handoffs in Snoop, where the soft state is replicated by

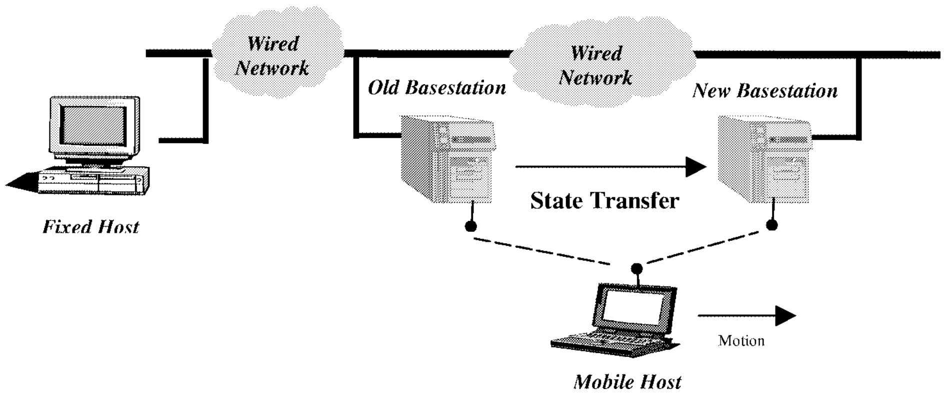

Fig. 6. Mobile performs an handoff from one basestation to another during the state transfer.

initiating a multicast when an handoff is *imminent* [11]. In our approach, we exchange state information between the basestations using unicast data transfer. Specifically, we rely on the lower-layers to provide an indication to the mobile that the handoff is imminent and the address of the candidate basestation. The state transfer of send/receive buffer is initiated once the previous basestation is notified. By the time the mobile actually decides to handoff, much of the data have already been transferred, as shown in Fig. 6. After the handoff, the remaining data and the state of TCP/MTCP variables are transferred to accurately mirror the previous state. In the event that the handoff does not occur, the candidate basestation discards the partial state information after some timeout. We believe that this technique can reduce the handoff latency, particularly in situations where send/receive buffers are large and need to be transferred over slow links.

### *3.6. Application re-linking*

One of the criticisms of 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 [34]. Our approach in solving this issue is to subsume the mechanisms used to create an indirect connection into MTCP, 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.

### *3.7. 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 and communicates with NDIS-compliant link-layer drivers [41] 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 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).

To further reduce protocol-processing complexity, we implemented a lightweight socket layer, as a dynamic link library (DLL), which bypasses various socket layer components of the Microsoft implementation. This lightweight socket layer directly interfaces with the MTCP protocol driver in the kernel to communicate requests made by applications. The resulting streamlined protocol stack is shown in Fig 7. We point out that our goal is not to advocate the creation of a new or incompatible socket interface, but rather to investigate the impact of streamlining on overall processing load. In Section 5, we present our performance results, which provide some useful insights.

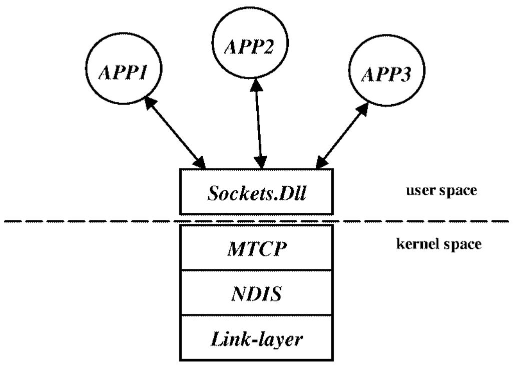

Fig. 7. MTCP protocol stack implementation.

Most TCP implementations, such as BSD, employ two types of timers: fast timer (200 ms) and slow timer (500 ms) [8]. 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 200 ms to process both delayed acknowledgments and retransmissions. Interestingly, our experiments indicate that timeout values on the Microsoft TCP (*MS*-TCP) implementation are more accurate than in the BSD implementation. We have seen timeout values in the range of 200–500 ms on *MS*-TCP, while most timeouts on BSD are typically multiples of 500 ms.

Finally, 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 [8].

## 4. Experimental setup and performance metrics

We performed various experiments to compare between the performance of MTCP and TCP in terms of the protocol processing overhead and the ability to recover from wireless losses. We chose to use TCP for our comparisons due to the lack of any other comprehensive solution that addresses *both* of these issues. In this section, we present our experimental testbed, error model, and performance metrics used to evaluate these protocols.

### 4.1. Experimental setup

Our experimental testbed is configured as a simple wireless LAN topology, as shown in Fig. 8. We chose to use this simple configuration in order to perform the experiments in a controlled setting and to eliminate any extraneous factors affecting protocol performance. Specifically, since there are no routers with the exception of the basestation, congestion is minimized or eliminated. In addition, the network is isolated, such that no other traffic flows over the testbed. The testbed consists of Pentium-based personal computers running the Windows 95 operating system. The Win95 machine in the center of the figure is configured to act as a basestation using the *WinRoute* software [42] to route packets to and from the mobile host. The fixed host and the basestation communicate over a 10 Mbps Ethernet link, while the basestation and the mobile host are connected to each other using either 1.6 Mbps Proxim RANGELAN2 [43] or 10 Mbps Ethernet. Proxim's RANGELAN2 is a wireless LAN product that operates at 2.4 GHz in the unlicensed ISM band. We chose to test the protocols under two different configurations to observe the effects of our error models on protocol performance at different data rates. Also, we note that Ethernet data rates are achievable in wireless technologies such as IEEE 802.11 or HIPERLAN [44]. The peak throughput for TCP bulk transfers is measured to be approximately 8.5 Mbps on Ethernet and 0.66 Mbps on Proxim RANGELAN2.

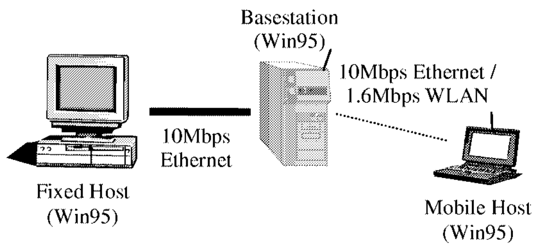

Fig. 8. Network configuration.

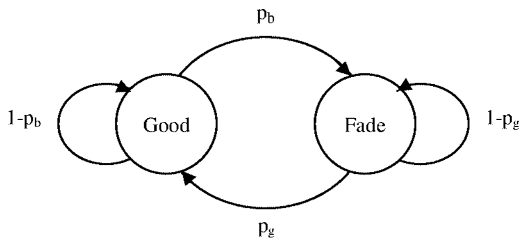

Fig. 9. Fading error model.

## 4.2. Error model

To understand and investigate the performance of different protocols under controlled error conditions, we simulate errors using a two-state Markovian burst-error model, as shown in Fig. 9. This model, which is based on the Elliot-Gilbert error-model [45,46], 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 *Fade* state. Once in the *Fade* 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 *Fade* state to the *Good* state. The state transition probabilities can be implemented either on a time-basis, bit-basis or packet-basis. We implement a temporal model using a distribution for fade duration and fade-free duration. Mean fade duration corresponds to the expected time spent in the *Fade* state. Similarly, mean fade-free duration corresponds to the expected time spent in the *Good* state. We use an exponentially distributed random variable to choose the duration of each state, around a particular mean duration. Depending on the state, all bits during that duration are either peremptorily counted as erroneous or error-free. A packet containing erroneous bits is actually corrupted by modifying the packet checksum.

Another parameter in the burst-error model is the fraction of the total time spent in the Fade state. We refer to this quantity as the *fade ratio*. Using fade ratio and mean fade-duration, which form the descriptive basis of the burst-error model, we can derive all other quantities including the mean fade-free duration and the state transition probabilities. Following equations can be applied (see Fig. 9 for definition of variables):

$$p_\mathrm{g} = 1/\mathrm{FD}, \text{ where FD is the fade duration,} \tag{1}$$

$$p_\mathrm{b} = \frac{\mathrm{FR}p_\mathrm{g}}{1-\mathrm{FR}}, \text{ where FR is the fade ratio,} \tag{2}$$

$$\text{Error-free duration} = 1/p_\mathrm{b}. \tag{3}$$

Furthermore, we derive the equations below to study the impact of our temporal error-model on packet losses. In doing so, we define another quantity called bit corruption rate (BCR), which refers to the ratio of bits corrupted. Since our model is time-based, it is possible that during certain period, bits are not corrupted even though channel fading occurs. (Note that we do not refer to BCR as bit error rate (BER), because BER is commonly interpreted as an un-correlated bit error rate, while fading errors are considered to be correlated.) We have observed that BCR is fairly close to the fade ratio when timeouts are rare.

$$\text{Packet Burst Loss Length} = 1 + \frac{\text{FD}}{S},\ \text{where } S \text{ is the mean packet size in time,} \tag{4}$$

$$\text{PER}/\text{BCR} = 1 + \frac{S}{\text{FD}},\ \text{where PER is packet error rate.} \tag{5}$$

An interesting observation to be made from the above equations is that packet burst-loss length decreases, while PER to BCR ratio increases with packet size. Since packet size in time is inversely proportional to the data rate, the above equations provide useful insights into understanding the effects of the burst-error model at different data rates. We exploit these observations to explain some experimental results presented in Section 5.

In our implementation, all packet errors are injected before being transmitted over the wireless link. Specifically, outgoing packets are intercepted below the transport protocol layer. Before being forwarded to the link-layer, each packet is subjected to the burst-error model and corrupted accordingly by modifying the checksum. Losses are generated in both directions of the wireless link, so acknowledgments are dropped as well.

### 4.3. *Performance metrics*

In accordance with the goals of our research, there are two principal criteria for evaluating protocol performance: (1) reduction in the protocol processing, and (2) improvement in protocol performance in the presence of wireless losses. We have designed two simple tests for obtaining performance data. The *unidirectional test* (also referred to as the bulk transfer test) transfers 100 000 socket buffers of 1460 bytes from the fixed host to the mobile host. All measurements for evaluating protocol performance in the presence of wireless losses are made using the unidirectional test because it exercises congestion control mechanisms and fast retransmissions. The *bidirectional test* ‘ping-pongs’ socket buffers between the sender and receiver. Unlike unidirectional test, this test is suitable for evaluating protocol processing, because each buffer is transferred in a separate packet. Furthermore, 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 [47] and the timing of delayed acknowledgments can cause a significant impact on the throughput of unidirectional transfers.

We evaluate protocol processing using the following metrics: *throughput*, *CPU usage*, and *protocol latency*. Protocol processing can be conceptually divided into various tasks, such as context switches (including user-to-kernel mode switch), memory copies, interrupt processing, and general protocol work. We evaluate each metric at different application-level buffer sizes to stress the protocol stack in different ways. Specifically, small packets increase the frequency of the interrupts, while large packets increase the number of bytes copied within the protocol stack. We define throughput as the ratio of the total number of received bytes (at the application layer) to the total transfer time. CPU-usage was measured using commercially available software, such as *System Monitor* and *Wintop* [48]. It indicates how much of the CPU is utilized on the average by a particular protocol. Comparing this quantity to the throughput achieved provides an insight into the amount of CPU energy consumed by the protocol. *Protocol latency* is defined as the average amount of time it takes for a buffer to traverse the protocol stack from the application layer to the link layer. This data is obtained by looping back the bidirectional test on a single machine; i.e., both ends of the connection reside on the same machine. The average latency is computed based on the amount of time it takes for some number of packets to be exchanged.

The second criterion for performance evaluation is addressed by metrics such as *throughput*, *mean inter-buffer delay*, and *inter-buffer delay-jitter*. Protocols are evaluated according to each metric at different packet error rates, fade ratios, and fade durations. Our performance graphs use normalized throughput, which is simply the ratio of the actual throughput to the peak throughput achieved by the quickest protocol (in this case MTCP) under ideal conditions.

We also evaluate the impact of packet losses on the delays experienced by applications in transferring data buffers. The importance of this study is to ascertain the performance available to real-time applications in lossy conditions. We introduce a performance metric called *inter-buffer delay*. The mean inter-buffer delay is the average waiting time that an application experiences before receiving the next data buffer. Since applications receive data in order, the inter-buffer delay is a useful indicator of how quickly the ‘holes’ in the receive-buffer are filled up and packets are delivered to the application. The *inter-buffer delay-jitter* is the standard deviation of the inter-buffer delay. It provides insights into variations in the end-to-end application-layer latencies in the presence of errors.

## 5. Performance evaluation

In this section, we present the results of our experiments comparing the performance of MTCP with TCP. Our comparisons are based on various performance metrics that were described in the previous section. In Section 5.1, we evaluate the protocols for processing efficiency. Section 5.2 presents our results comparing the protocols at different packet error rates and fade ratios.

### 5.1. Protocol processing

In this section, we compare the processing overhead of MTCP and TCP/IP using various metrics, such as peak bi-directional throughput, CPU-usage, and protocol latency. We also study the impact of the payload on protocol processing by varying the buffer size for each metric. Most tests were conducted in the ethernet setup in order to stress the protocols. Note that in this section, we aim to compare the MTCP and the TCP/IP protocol stacks and not just the two protocols (MTCP and TCP).

Figure 10 compares the peak bi-directional throughput of MTCP and TCP transfers at different buffer (payload) sizes. The results show that MTCP achieves an improvement of about 40% at 100 bytes exchanges and about 12% at 1460 bytes exchanges. At small buffer sizes, packets need to be processed at a faster rate, since the transmission time is relatively small. Substantial improvements at small buffer sizes indicate that MTCP is able to utilize the CPU more efficiently than TCP. As the buffer size is increased, the packet transmission time increases and processing optimizations contribute less to the improvement in throughput. Therefore, we see a steady decline in the improvement curve with an increase in buffer size. We have also verified that the throughput improvements were not primarily due to the reduction in the header size. To verify this, we temporarily increased the MTCP header size by 24 bytes. The results are also shown in Fig. 10 using the curve labeled MTCP_24. The differences between MTCP_24 and MTCP are less than 5%.

Figure 11 plots the average CPU-usage over the lifetime of the connection for each protocol stack. We note that the fluctuations in the CPU-usage were minor during the data transfer. According to the results, TCP/IP utilizes

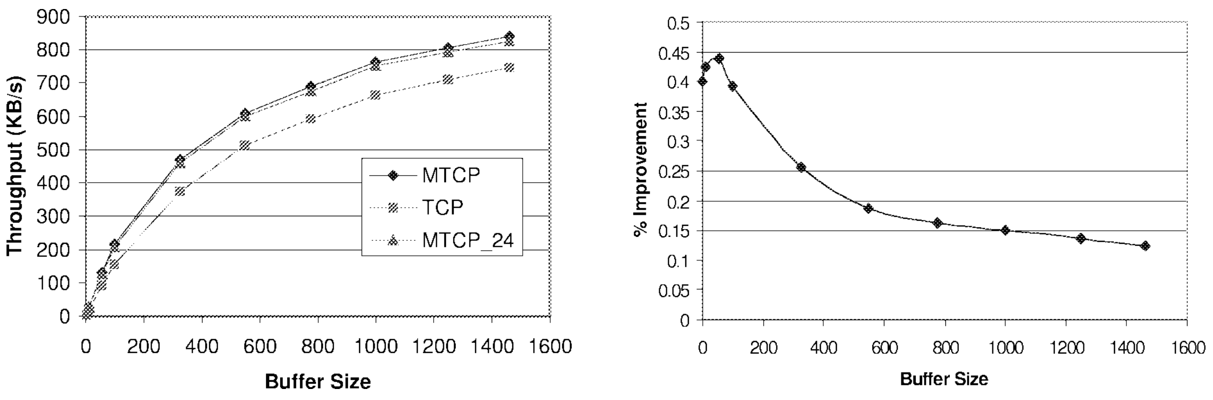

Fig. 10. Comparison of bi-directional throughput in MTCP and TCP.

substantially greater amount of CPU resources than MTCP. Specifically, CPU-usage of MTCP is approximately half of TCP/IP regardless of the buffer size. A metric expressed as average CPU-usage divided by the throughput serves as an indicator of the total CPU-energy consumed per kbyte of data transferred. This metric, shown in Fig. 12, indicates that TCP consumes 100–200% more CPU-energy per kbyte of data than MTCP in the Ethernet setup.

We now compare the two protocol stacks according to protocol latency, which is the time required for a packet to traverse the protocol stack from the application layer to the link layer. This metric is interesting because it focuses on the actual processing overhead of a protocol, eliminating the packet transmission time. Figure 13 shows protocol latency of MTCP and TCP/IP at different buffer sizes. We observe that the protocol latency increases almost linearly with the buffer size. The dotted lines labeled MTCP-fit and TCP/IP-fit are curves fitted to the experimentally obtained points. The results indicate that the protocol latency in MTCP is much lower than TCP/IP. The intercept of the MTCP curve is 34 $\mu$s, while it is 123 $\mu$s for TCP/IP. This is the difference when the buffer size is virtually zero, indicating the difference in the basic processing required for each buffer regardless of its size. Furthermore, the slope of the curve is 0.034 $\mu$s/byte for MTCP and 0.092 $\mu$s/byte for TCP/IP. This suggests that the MTCP stack performs fewer memory copies of a buffer before handing it over to the link-layer.

All the metrics examined so far provide an indication of the *overall* protocol processing overhead. Using a profiling tool, called VTune [49], we were able to determine the percentage of CPU time spent at different protocol layers. In addition, the tool helped us to identify the individual software components that form the TCP/IP stack. Figure 14 shows the percent of CPU utilized in different parts of the TCP/IP stack. Note that the figure does not include the CPU time spent in the other parts of the operating system on behalf of protocol processing, since this was practically difficult to determine. The results reveal that a significant amount of CPU time is utilized in the

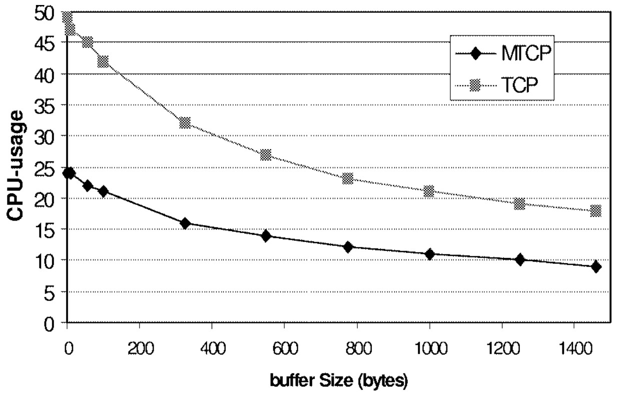

Fig. 11. CPU utilization of MTCP and TCP.

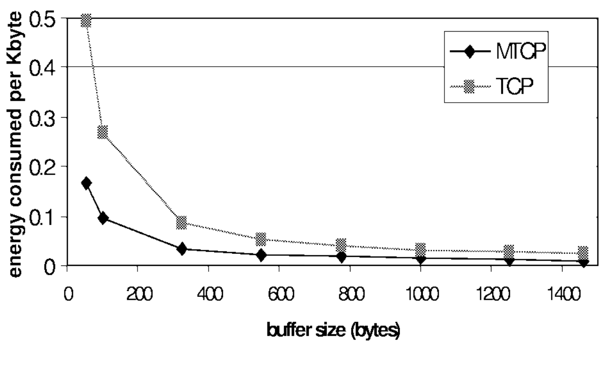

Fig. 12. Energy consumption per kbyte of MTCP and TCP.

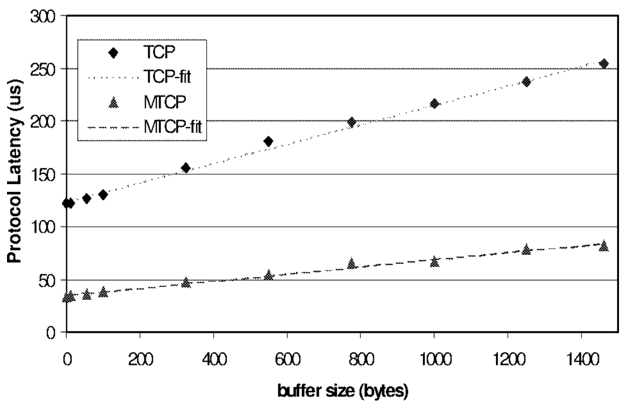

Fig. 13. Protocol stack latency of MTCP and TCP/IP.

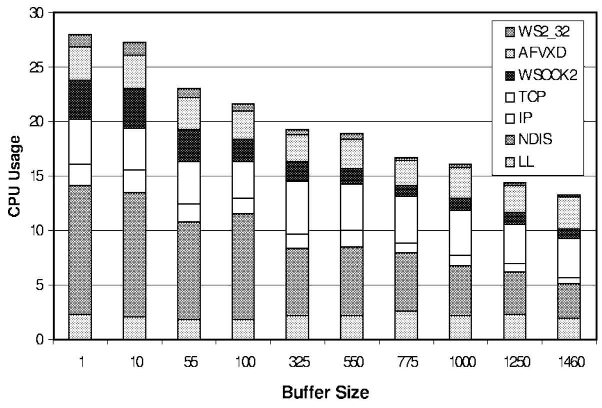

Fig. 14. CPU usage in different components of TCP/IP stack.

socket layer components (WS2_32, AFVXD, WSOCK2). On the other hand, the lightweight socket library that we implemented for MTCP utilized less than 2% of the CPU (not shown here). Thus, it is recommended to minimize the complexity of the socket layer as far as possible. Another interesting finding is that the percentage of the TCP and IP layer (excluding other components of TCP/IP) used by IP ranges from 15% to 33%. This suggests the kind of improvement possible by eliminating the IP layer from the path of the protocol processing.

### 5.2. *Wireless losses*

We conducted our experiments on a wireless LAN testbed, shown in Fig. 8. The testbed had two variations. Specifically, the wireless segment was tested using both wireless (0.66 Mbps) and Ethernet (8.5 Mbps) technologies. In our discussion, we refer to the two variations as *wireless setup* and *Ethernet setup*, respectively. In our first experiment, the fade ratio was varied between 0.1% to 10% and the mean fade duration was 1250 wireless-bytes, corresponding to 17 ms. This resulted in the approximate packet error rate ranging from 0.2% to 22%, as predicted by Eq. (5). We verified the choice of the fade duration using a previous study. In [50], the author presents an experimentally obtained CDF of error-duration over a wireless network. This curve approximates the CDF of an exponential distribution and has a mean of about 18 ms. The maximum possible window size for the connection was 16 kbytes and the socket buffer size was 1460 bytes.

Under ideal (loss-free) conditions, the improvement achieved by MTCP over TCP is about 8%. This base improvement can be attributed to a combination of reasons, such as smaller packet headers, reduction in processing

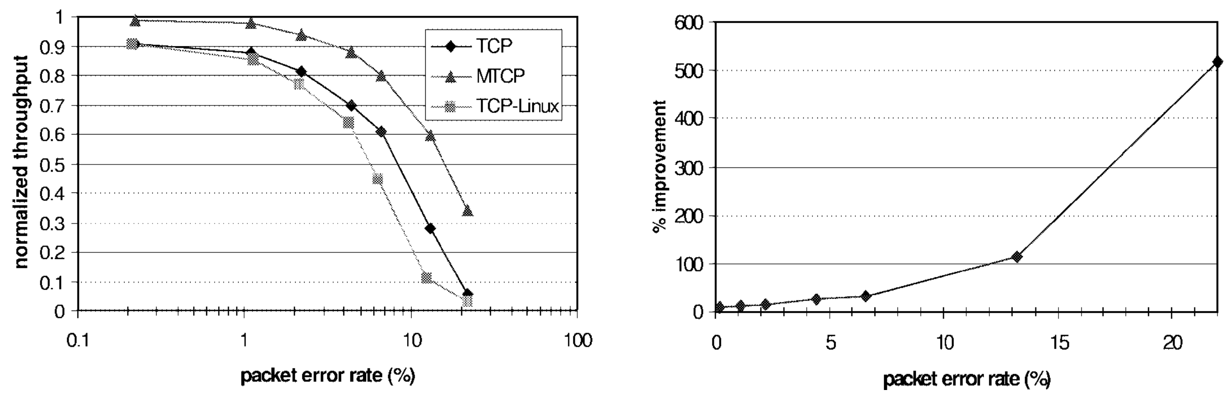

Fig. 15. Comparison of protocols in the wireless setup.

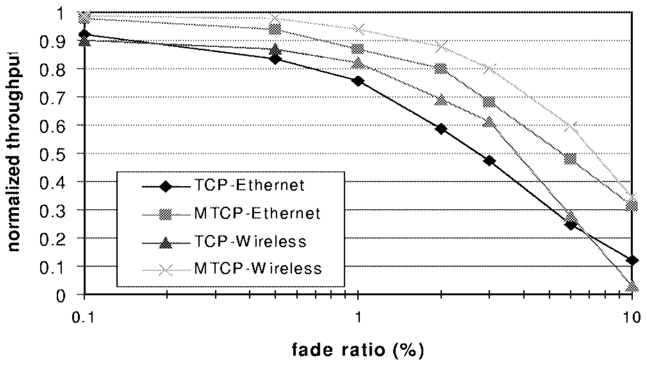

Fig. 16. Comparison of TCP and MTCP at different data rates.

overhead, elimination of slow-start, and the small differences in the timing of ACKs. Figure 15 compares the performance of MTCP and TCP at various packet error rates in the wireless configuration. We observe that for packet error rates of over 10%, MTCP performs significantly better than TCP. For instance, the throughput improvement is approximately 75% at 10% packet error rate, and more than 400% at 20% packet error rate. At lower packet error rates (less than 5%), the results indicate that TCP is quite robust in dealing with losses. As the packet error rate increases, burst losses become more frequent. In such situations, a burst loss typically occurs even before congestion-window has a chance to sufficiently expand after the previous loss. This increases the occurrence of coarse-timeouts and causes substantial degradation of throughput. Besides eliminating congestion control mechanisms, MTCP improves upon TCP by employing selective acknowledgments (SACKs). The benefit of using SACKs is that MTCP is able to recover from losses much quicker than TCP. In addition, SACKs further reduce the possibility of timeouts, because small windows are generally enough to detect all packet losses. During our experiments, we, in fact, noticed that some of the TCP transfers failed to complete at packet error rates of over 15%. On the other hand, MTCP was robust enough to finish transfers at packet error rates greater than 50%.

Figure 15 also shows the performance of the TCP implementation of Linux (labeled as TCP-Linux). The results indicate that TCP-Linux performs worse than the Microsoft implementation. We believe that this is because of coarser timer granularities in TCP-Linux. As mentioned earlier, standard BSD implementations use timeout values as multiples of 500 ms, while we have observed finer settings in both TCP-Linux and *MS*-TCP. Thus, we expect higher improvements in comparison to other implementations. Figure 16 compares TCP and MTCP in the wireless and the Ethernet setup at different fade ratios. The mean fade duration is normalized to 1250 wireless-bytes, as before. This corresponds to approximately 17 ms. We make two observations based on the figure. First, the nor-

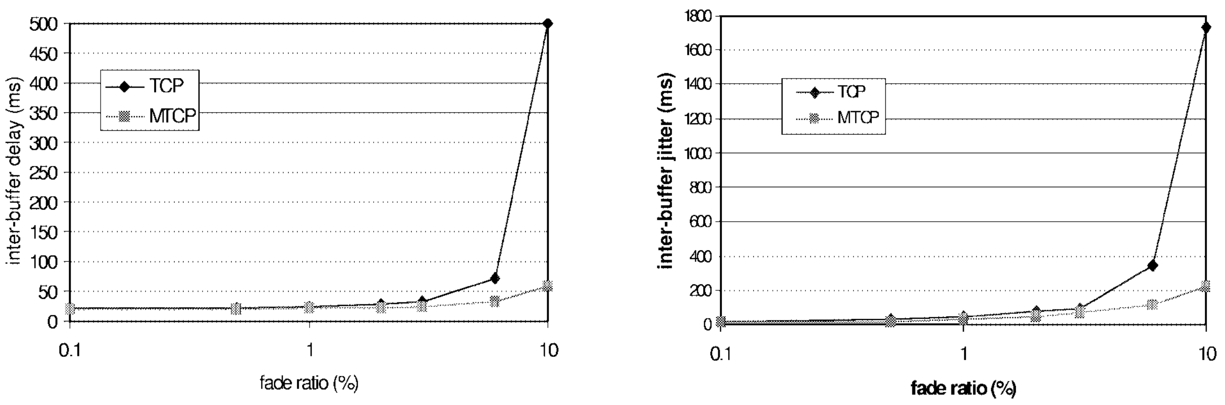

Fig. 17. Mean inter-buffer delay and jitter in the wireless case.

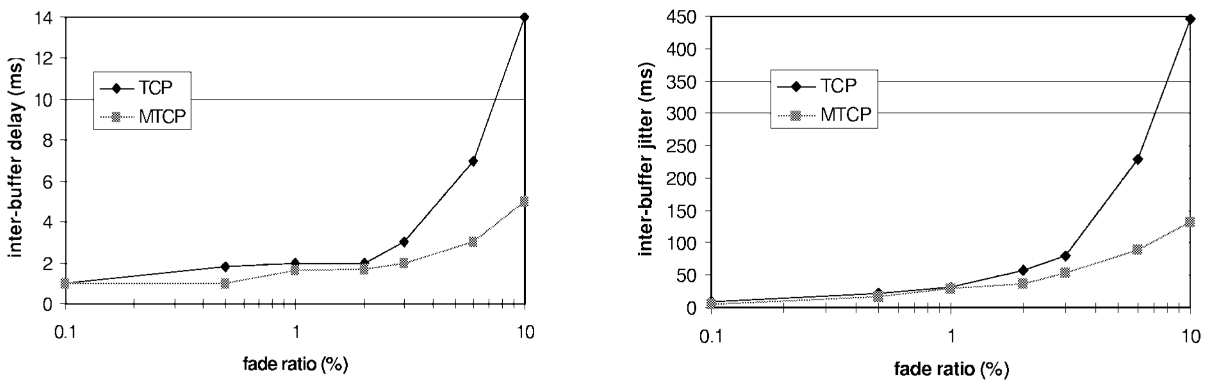

Fig. 18. Mean inter-buffer delay and jitter in the Ethernet case.

malized throughput for both protocols is less at Ethernet speed as the fade ratio increases. Second, the normalized throughput in the wireless setup sharply declines at 3% fade ratio and crosses the Ethernet curve at 10% fade ratio.

We can explain these observations based on two key points. First, the burst packet loss length increases with the transmission rate, as packets get shorter in time. And second, the total number of fades is less at higher data rates, since the mean number of fades per unit-time is the same regardless of the data rate. The first observation that both the protocols perform poorly in the Ethernet case is the result of a greater packet burst loss length. A long burst loss is much more likely to cause a coarse-timeout. The mean burst loss length is estimated to be about 12 packets [Eq. (4)], as opposed 1.8 packets in the wireless case. This constitutes a loss of more than an entire window of packets (recall, the maximum window size is 16 kbytes). Since timer granularities ($\sim$200–500 ms) are coarser than the typical round-trip time ($<$50 ms), timeouts have a greater impact in reducing the *normalized* throughput at higher data rates, as they constitute a larger share of the total time to transfer some number of packets. MTCP does not improve much over TCP in the Ethernet case, because it also suffers from a higher frequency of timeouts. However, MTCP recovers slightly quicker than TCP after timeouts, as it does not perform congestion control. As the fade ratio increases, there is an increase in the frequency of fades. This seems to have a greater impact on the wireless case where the number of fades is higher and the fades are much shorter relative to the packet size, causing multiple but sparse packet losses within a window. As a result, protocols in wireless case are increasingly prone to expensive timeouts, attributing to a sharp decline at higher fade ratios.

We now turn to presenting the impact of packet losses on the delays experienced by applications in exchanging data buffers. Figures 17 and 18 show the inter-buffer delay and jitter in the *wireless* and the *Ethernet* cases respectively at different fade ratios. As expected, the results indicate that the mean inter-buffer delays are much lower in MTCP than TCP in both the cases. Inter-buffer jitter is also smaller in MTCP, indicating that it is consistently efficient at recovering from errors. Interestingly, the jitter in the *Ethernet* case is almost as large as the jitter in the *wireless* case at each fade ratio, even though the mean inter-buffer delay is much smaller in comparison. This is due to the increased number of timeouts in the *Ethernet* case, contributing to large fluctuations in the buffer waiting time.

## 6. Design alternatives

As we mentioned earlier, many ideas proposed in MTCP are not restricted to the split-connection model. In this section, we briefly offer some design alternatives that can incorporate the key features of MTCP, namely the minimization of processing overhead and the reduction of wirelessly transmitted data.

One approach is to design a modified TCP on the mobile host, which uses the IP layer only for connection control (establishment/teardown) and bypasses it for all subsequent data packets. Unlike the split connection approach, the modified TCP communicates with the fixed host TCP without splitting the connection. To bypass IP and compress the IP addresses and TCP port numbers, we introduce an agent at the basestation, which is similar to the Snoop agent. During the three-way connection establishment handshake, the agent negotiates unique forward and reverse connection identifiers (CIDs) with the modified TCP, while piggybacking on the TCP connection packets. Once the CIDs are established, all subsequent data packets on the wireless segment carry a compressed header and the modified TCP bypasses IP processing. The agent's function is to compress packets heading toward mobile device and decompress packets destined for the fixed host. Like in Snoop, the agent and the modified TCP communicate with each other to perform TCP-aware error recovery over the wireless segment. When an handoff occurs, the CIDs have to be renegotiated either using special control packets or by piggybacking on TCP packets. We note that in addition to compressing the IP addresses and TCP port numbers, one can also use a low loss compression technique, as in [24], to compress other TCP/IP header entries.

Regardless of the approach, it is also possible to streamline the upper-layers of protocol stack. Specifically, the design of the socket layer can be greatly simplified by reducing processing overhead on the mobile device, as we have done in our implementation.

## 7. Future work

We are planning to incorporate asymmetric protocol design techniques [4] 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. We are also working on improving the efficiency and accuracy of our timer implementation.

While transport protocols are more aware of end-to-end application requirements, link-layer protocols have the advantage of knowing link-specific attributes, such as hardware status. We would like to investigate the efficacy of designing transport protocols that adapt to the conditions at the link-layer, in order to minimize wasteful transmissions when the link quality is bad. Such a link-aware transport solution can be possible by propagating information available at lower-layers through the protocol stack. We also intend to extend our performance results to show the impact of reduced processing and communication overhead on the power-saving features and overall consumption of battery energy.

## 8. Summary and concluding remarks

In this paper, we presented the design and performance of a lightweight protocol architecture for the transport-layer, which significantly reduces communication load on the mobile device and minimizes the use of the wireless medium. Our implementation employs the proxy-style of communication provided by the split-connection model to develop a new streamlined protocol, MTCP, over the wireless segment. Since the wireless segment is a single link, many functions typically required for a transport protocol can be either eliminated or simplified. Specifically, we are able to eliminate the IP layer from the path of protocol processing, as all packets are routed through the basestation. By doing so, we also reduce the size of the packet header. Furthermore, MTCP uses a simple technique to compress the IP addresses and the TCP port numbers, while subsuming the addressing information into its own header. Since all losses on the wireless segment are non-congestion-related, the design of MTCP does not include any congestion control mechanisms. Furthermore, the protocol implementation streamlines the socket layer to investigate its impact on protocol processing overhead.

Our experimental results indicate that MTCP is much more efficient than TCP in terms of protocol processing. For instance, the results show that the MTCP processing time per *packet* is approximately only one fourth that of TCP, while the processing time per *byte* is only about one third that of TCP. Using MTCP, the utilization of the CPU is reduced by up to 50% and the overall CPU energy is conserved by a factor of one third, depending on the message size and the data rate of the connection. The results show that IP processing constitutes between 15 to 33 % of the TCP and the IP processing overhead. In addition, the results demonstrate that protocol processing overhead can be considerably reduced by simplifying the socket layer.

The design of MTCP also includes various robust loss recovery techniques to combat burst losses on the wireless link. Most of the techniques take advantage of the fact that all packets arrive in sequence over the wireless segment. The simulation results show that MTCP substantially improves over TCP. For instance, in the wireless setup, the throughput achieved by MTCP is 400% higher than TCP at a packet error rate of 20%. Our performance measurements under identical lossy conditions indicate that the normalized throughput depends on the data rate of the connection. Furthermore, the results indicate that the delay-jitter is considerably lower in MTCP, suggesting that the protocol is consistently effective in recovering from packet losses.

Finally, we note that the design of MTCP is not confined to the split-connection model. In this paper, we described some alternative approaches that can be used to implement the key ideas of MTCP.

## 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.R. Lorch and A.J. Smith, Software strategies for portable computer energy management, *IEEE Personal Communications*, June, 1998.

[4] Z.J. Haas, Mobile-TCP: an asymmetric transport protocol design for mobile systems, in: *ICC '97*, Montreal, Canada, 1997.

[5] 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*, 1993.

[6] 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*, 1994.

[7] J.B. Postel, Transmission control protocol, RFC, Information Sciences Institute, Marina del Rey, CA, RFC-793, September, 1981.

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

[9] R. Caceres and L. Iftode, Improving the performance of reliable transport protocols in mobile computing environments, *IEEE JSAC* **13**(5) (1994).

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

[11] H. Balakrishnan, S. Seshan and R.H. Katz, Improving reliable transport and handoff performance in cellular wireless networks, *Wireless Networks* **1**(4) (1995), 469–481.

[12] K. Brown and S. Singh, M-TCP: TCP for mobile cellular networks, *Computer Communications Review* **27**(5) (1997).

[13] E. Chan, K. Govil and H. Wasserman, Comparing algorithms for dynamic speed-setting of a low power CPU, in: *Proc. First ACM Int'l Conf. On Mobile Computing and Networking (MOBICOM '95)*, Berkeley, CA, 1995, pp. 13–25.

[14] M. Weiser et al., Scheduling for reduced CPU energy, in: *Proc. First USENIX Symp. on Operating Sys. Design and Implementation*, Monterey, CA, 1994, pp. 13–23.

[15] J. Lorch and A.J. Smith, Reducing processor power consumption by improving processor time management in a single-user operating system, in: *Proc. Second ACM Int'l Conf. On Mobile Computing and Networking (MOBICOM '96)*, Rye Brook, NY, 1996, pp. 143–154.

[16] M.B. Srivastava, A.P. Chandrakasan and R.W. Brodersen, Predicive system shutdown and other architectural techniques for energy efficient programmable computation, *IEEE Trans. on Very Large Scale Integration (VLSI) Sys.* **4**(1) (1996), 42–55.

[17] N. Suzuki and S. Uno, Information processing system having power saving control of the processor clock, United States Patent No. 5,189,647, Feb., 1993.

[18] J. Conger, *Windows API Bible*, Waite Group Press, Corte Madera, CA, 1992.

[19] M. Pietrek, *Windows Internals*, Addison Wesley, Reading, MA, 1993.

[20] H. Woesner, J. Ebert, M. Schlager and A. Wolisz, Power-saving mechanisms in emerging standards for wireless LANs: the MAC level perspective, *IEEE Personal Communications*, June, 1998.

[21] M. Zorzi and R.R. Rao, Error control and energy consumption in communications for nomadic computing, *IEEE Trans. on Computers* **46**(3) (1997), 279–289.

[22] P. Lietierri, C. Fragoulli and M.B. Srivastava, Low power error control for wireless links, in: *Proc. MobiCom '97*, Budapest, Hungary, 1997, pp. 139–150.

[23] E. Ayanoglu, S. Paul, T.F. LaPorta, K.K. Sabnani and R.D. Gitlin, AIRMAIL: a link-layer protocol for wireless networks, *ACM/Baltzer Wireless Networks Journal* **1**(Feb.) (1995), 47–60.

[24] M. Degermark et al., Low-loss TCP/IP header compression for wireless networks, in: *Proc. Second ACM Int'l Conf. On Mobile Computing and Networking (MOBICOM '96)*, Rye Brook, NY, 1996, pp. 1–14.

[25] V. Jacobson, Compressing TCP/IP headers for low-speed serial links, RFC 1144.

[26] P. Bellanger and W. Diepstraten, 802.11 MAC entity: MAC basic access mechanism/privacy and access control. Presented to IEEE P802, Mar., 1996.

[27] W. Mangione-Smith et al., A low power architecture for wireless multimedia sys.: lessons learned from building a power hog, in: *1996 Int'l Symp. On Low Power Electronics and Design Digest of Technical Papers*, Monterey, CA, 1996, pp. 23–28.

[28] B. Mangione-Smith, Low power communications protocols: paging and beyond, in: *1995 IEEE Symp. On Low Power Electronics Digest of Technical Papers*, San Jose, CA, 1995, pp. 8–11.

[29] D. Cheriton and C. Williamson, VMTP as the transport layer for high-performance distributed systems, *IEEE Commun. Mag.*, June, 1989, 37–44.

[30] S. Chanson, K. Ravindran and J. Robinson, The design and tuning of a transport protocol for local area networks, in: *Proceedings, IEEE INFOCOM '88*, 1988.

[31] K. Sabnani and A. Netravali, A high-speed transport protocol for datagram/virtual circuits, in: *Proceedings, SIGCOMM '89 Symposium*, 1989.

[32] R.W. Watson and S.A. Mamrak, Gaining efficiency in transport services by appropriate design and implementation choices, *ACM Transactions on Computer Systems* **5**(2) (1987), 97–120.

[33] W.T. Strayer, B.J. Dempsey and A.C. Weaver, *Xtp: The Express Transfer Protocol*, Addison-Wesley, Reading, MA, 1992.

[34] H. Balakrishnan, V. Padmanabhan, S. Seshan and R.H. Katz, A comparison of mechanisms for improving TCP performance over wireless links, in: *Proceedings of ACM SIGCOMM 96*, 1996.

[35] A. DeSimone, M.C. Chuah and O.C. Yue, Throughput performance of transport layer protocols over wireless LANs, in: *Proc. IEEE GLOBECOM '93*, Houston, 1993, pp. 542–549.

[36] D. Maltz and P. Bhagwat, MSOCKS: An architecture for transport layer mobility, in: *Proc. INFOCOM*, 1998.

[37] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, TCP selective acknowledgment options, RFC-2018, 1996.

[38] S. Keshav and S. Morgan, SMART retransmission: performance with overload and random losses, in: *Proc. Infocom '97*, 1997.

[39] C. Perkins, ed., IP mobility support, Internet Engineering Task Force, Internet Draft, draft-ietf-mobileip-protocol-15.txt, 9 Feb., 1996.

[40] R. Caceres and V. Padmanabhan, Fast and scalable wireless handoffs in support of mobile internet audio, *Baltzer Journals*, **13**(Nov.) (1997).

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

[42] Winroute Homepage. http://www.winroute.com (1999).

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

[44] ETSI, High performance radio local area network (HIPERLAN), Draft Standard ETS 300 652, Mar., 1996.

[45] N. Gilbert, Capacity of a burst noise channel, *Bell Sys. Tech J.* **39**(Sept.) (1960), 1253–1266.

[46] O. Elliot, Estimates of error rates for codes on burst-noise channels, *Bell Sys. Tech J.* **42**(Sept.) (1963), 1977–1997.

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

[48] Microsoft Homepage, http://www.microsoft.com (1999).

[49] VTune Homepage, http://www.intel.com/vtune (1999).

[50] H. Balakrishnan, PhD thesis, University of California at Berkeley, 1998.