ELSEVIER

Computer Networks 40 (2002) 515–535

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

On experiments with a transport protocol for pervasive computing environments

Sasikanth Avancha *, Vladimir Korolev, Anupam Joshi, Timothy Finin, Yelena Yesha

Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Cirle, Baltimore, MD 21250, USA

Received 27 June 2001; received in revised form 15 March 2002; accepted 30 April 2002

Responsible Editor: G. Morabito

Abstract

It is well-known that TCP performs poorly in a wireless environment. This paper presents an empirical performance analysis of TCP on cellular digital packet data, Bluetooth and wireless LAN. We also present CentaurusComm, a message based transport protocol designed to perform well in low bandwidth networks and resource poor devices. In particular, CentaurusComm is optimized to handle data exchanges consisting of short message sizes. The application used to perform all the experiments is typical of common applications that would use these protocols and network technologies. Typical mobile devices used in the experiments included Palm Pilots. We present results of performance evaluation of TCP and CentaurusComm on different wireless technologies.

Keywords: CentaurusComm; CDPD; Bluetooth; IEEE 802.11b; Empirical analysis; Wireless networks; Transport protocols

1. Introduction

Wireless networks of the present and future are envisioned to range from body area networks to satellite wide area networks (WANs). These will include Bluetooth [1] based systems, 802.11 based wireless LANs (WLANs) [2] and WANs based on packet radio technologies like cellular digital packet data (CDPD) [22] and general packet radio service (GPRS). [3] At an abstract level, data exchange in wireless networks is very similar to that in wired networks, except for smaller data sizes. Connection-less and connection-oriented data transfer mechanisms exist in most wireless systems. The amount of data exchanged is typically of the order of hundreds of bytes. Maximum transfer units (MTU) specified by kernels optimized for wireless networks also tend to be of the order of hundreds of bytes for typical applications. TCP

* Corresponding author. Tel.: +1-410-455-2861.

E-mail addresses: savanc1@csee.umbc.edu (S. Avancha), vkorol1@csee.umbc.edu (V. Korolev), joshi@csee.umbc.edu (A. Joshi), finin@csee.umbc.edu (T. Finin), yeyesha@csee.umbc.edu (Y. Yesha).

[1] http://www.bluetooth.com.

[2] http://www.ieee.org/groups/802/11.

[3] http://www.gsmworld.com.

1389-1286/02/$ - see front matter
PII: S1389-1286(02)00294-3

has been the protocol of choice for reliable, connection-oriented data transfer on wired networks. Adapting TCP to wireless networks has thus become an important area of research.

TCP performance has been extensively researched on wired networks that have high bandwidth and throughput, and low latency and delays [2,9,10,16]. As expected, TCP performs very well on wired networks. However, research on TCP performance over wireless networks has shown that it fails under certain conditions. Non-congestion losses (losses due to wireless channel errors or client mobility) mostly contribute to the poor performance of TCP. This is because TCP implicitly assumes that all losses are due to congestion and reduces the window on the sender [17]. If the losses are not due to congestion, then TCP unnecessarily reduces throughput leading to poor performance.

We propose a new protocol called CentaurusComm that provides reliable, message oriented data transmission. This protocol has been designed to cater to resource poor devices exchanging small amounts of data in low bandwidth networks. We evaluate the performance of this protocol and show that, under certain conditions, it performs better than TCP. In Section 3, we explain the reasons for proposing a new protocol as opposed to making further modifications to TCP.

A thorough quantitative analysis of TCP and CentaurusComm performance on different types of wireless networks like CDPD, WLAN and Bluetooth is essential in order to understand their behavior on each type of wireless network. It is possible to analyze the performance of TCP and CentaurusComm by simulating these different networks and associated environments using simulators like ns-2 [1]. Such simulations can provide very accurate information on the behavior of such protocols. We believe that the main drawback of such analyses is that the effects of uncontrolled parameters such as signal strength, channel error rates, channel "busyness" and noise cannot be accurately studied. The reason for this is that it is difficult to model these parameters in simulators. For example, error rates can be modeled in ns-2 using the two-state Markov model with a bit error rate of 10^{-6} signifying that the channel state is "good" and a bit error rate of 10^{-2} signifying that it is "bad". The problem with this kind of modeling is that these two values may never be seen during an actual transmission on a CDPD network. Error rates might actually vary between a certain range around these two fixed values. If, due to these variations in error rates, the channel state cannot be characterized as "good" or "bad", then TCP performance cannot be accurately studied or explained. Of course, it is possible to design a more complex simulator that uses an algorithm that could model error rates with greater accuracy. We believe that direct measurement is a much simpler and more accurate method of analyzing and studying TCP performance in wireless networks.

We now present a brief overview of the three wireless network technologies considered in this paper. CDPD is a packet switched communications network based on TCP/IP that normally operates as an overlay on top of the existing advanced mobile phone services (AMPS) infrastructure. It is in fact a digital cellular system designed for data transport that can operate independently or on any cellular system that uses 30 kHz channelization (e.g. AMPS analog systems in North America). CDPD is a representative wireless WAN (WWAN) that provides data rates of up to 19.2 Kbps. Bluetooth is a fast emerging wireless technology that provides short range, moderate bandwidth connections. It operates in the globally available 2.4 GHz ISM frequency band and provides data rates of up to 432 Kbps (symmetric) and 721 Kbps (unsymmetric). The Bluetooth specification has two power levels defined; a lower power level that covers the shorter personal area within a room, and a higher power level that can cover a medium range, such as within a home. It supports both point-to-point and point-to-multipoint connections. With the current specification, up to seven 'slave' devices can be set to communicate with a 'master' radio in one device. Several of these 'piconets' can be established and linked together in ad hoc 'scatternets' to allow communication among continually flexible configurations. The WLAN technology also utilizes the 2.4 GHz ISM frequency band. To communicate over this band, the WLAN uses the IEEE 802.11b standard. This standard covers two aspects of the WLAN protocol: media

access control layer (MAC) and physical transport layer (PHY). The MAC layer uses the carrier sense multiple access with collision avoidance (CSMA/CA) as a method of access. The PHY layer uses either frequency hopping or direct sequence spread spectrum (FHSS/DSSS) to transmit. Typical modes of connectivity in a WLAN are "ad hoc" and "infrastructure". The ad hoc mode allows peer-to-peer communication without any wired support. The infrastructure mode allows devices to communicate over the air with an access point (AP) that is connected to the wired network. The current WLAN, based on the IEEE 802.11b, provides data rates of up to 11 Mbps.

The rest of the paper is organized as follows: Section 2 discusses some well-known solutions for improving TCP performance over wireless networks and also describes prior work on empirical performance measurement of TCP. We describe the CentaurusComm protocol in detail in Section 3. Section 4 describes in detail, the performance metrics for TCP and CentaurusComm, and the measured CDPD parameters. This section also details the experimental setup for measuring TCP and CentaurusComm performance over CDPD, WLAN and Bluetooth. Section 5 details and analyzes the experimental results. Section 6 presents our conclusions and describes future work.

2. Prior work on improving TCP performance in wireless networks

There are two classes of solutions to the TCP performance problem in wireless networks. The problem can be solved either by modifying TCP, e.g. Indirect-TCP (I-TCP), or by replacing it with a protocol that is optimized for wireless networks, e.g. CentaurusComm.

Most of the prior research work has focused on making the TCP/IP stack smarter by modifying TCP. Proposed solutions either involve violating end-to-end semantics [3,7], modifying TCP code on the mobile client, the wired source or both [4,8] or introducing TCP-aware smarts in the base station [5]. Other solutions that attempt to improve TCP performance while retaining the end-to-end semantics include [19] and [20].

I-TCP was introduced in [3]; it proposed that the TCP connection be split at the wired-wireless network border, thus maintaining two connections—one over the wired network and another over the wireless network. Thus, the TCP on the wired host is unaware of non-congestion related losses on the wireless part of the connection. Another protocol similar to I-TCP, called MTCP was proposed in [7]. The main difference between I-TCP and MTCP is that, in the latter, the last byte of TCP is acknowledged to the wired host only after the mobile client receives it.

The explicit bad state notification (EBSN) protocol in [4] allows the base station to perform retransmissions locally and causes it to send a message to the wired host to prevent the latter from unnecessarily timing out. In the fast-retransmission approach [8], which is useful mainly during hand-offs, the mobile client generates a certain number of duplicate acknowledgments that cause the wired host to retransmit the lost segment(s) without waiting for the timeout period to expire. The Snoop protocol in [5] specifies that the base station should detect segment losses and locally retransmit the segment until the mobile client receives it.

The solution proposed in [20] is to modify the design of congestion control and reliability in TCP. Congestion control is done at the receiver using rate-based and inter-packet delay based mechanisms. Thus, the sender modifies the transmission rate (increase, maintain or decrease) based on the observations of the receiver. Reliability in [20] eliminates the need for a retransmission timeout. Based on information received in ACKs, the sender decides whether or not a retransmission is required. This protocol was proposed specifically to improve TCP performance over the CDPD network. In this work real-time performance measurements have been done but the authors warn against using the results to quantify TCP performance, because the experiments were performed in uncontrolled conditions.

In [19], the WTCP protocol resides on the base station alongside TCP. Data received by TCP is processed by WTCP. On receipt of data from a wired host, WTCP buffers the segment and timestamps it. This segment is transmitted to the

mobile host based on sequence number and window information present on the base station. Out-of-order segments are simply buffered at the base station and transmitted when appropriate. ACKs from the mobile host are processed at the base station and corresponding ACKs are sent to the wired host. Thus, end-to-end semantics are maintained in this protocol.

It may also be possible to improve TCP performance by monitoring real-time parameters indicating the state of the wireless connection (between the mobile client and base station) on the mobile client. Based on statistical computations on these parameters, the client could make decisions that would affect TCP on the wired host. For example, the TCP stack on the client in a CDPD network can be modified to defer transmission if it observes that the signal strength is close to a level that may indicate potential loss of data. In Freeze-TCP [21] the mobile client monitors the signal strength to anticipate an impending handoff. If it detects one, it advertises a window size of zero to switch the sender to a persist mode and prevent it from reducing the size of its congestion window.

As another example, it may be possible for the client to detect that the cell in which it is currently is busy 90% of the time and thus decide to defer transmission. Similarly, it may be possible to modify the TCP on the wired host to use some of the statistics, like the number of packets dropped or received in error, gathered by various entities in the system to make decisions regarding transmissions and receptions. Another solution would be to transfer responsibility directly to the application which would "peek" at some current statistics gathered by the lower layers to make transmission related decisions.

Work related to empirical TCP performance measurement over some wireless networks has been done and described in [5,13,18]. In [18], throughput was chosen as the performance metric and location dependent performance was measured and analyzed. WLANs have location dependent characteristics, therefore the variation in TCP throughput based on different locations of the mobile client was studied. In [13], interaction between TCP and radio link protocol (RLP) (GSM network specific link layer protocol) was the basis of performance analysis. TCP's utilization of the bandwidth the RLP provides to it, was the primary performance metric. This work concludes that link layer solutions can alone solve the problem of poor TCP performance. Various solutions discussed above were actually implemented and TCP performance was evaluated over a WLAN in [5]. This work also concludes that a reliable, TCP-aware link layer provides very good performance.

3. CentaurusComm transport protocol

We now discuss a solution that seeks to replace TCP on the "wireless gateways" with a protocol better suited to wireless networks. One of the main motivations for designing such solutions is the set of typical applications that use them. Mobile and wireless users typically run applications like web browsers, FTP clients and email clients on their devices. These applications mostly generate relatively short messages as opposed to continuous streams of data. Our approach described here is optimized to handle short messages more efficiently than TCP. The CentaurusComm transport protocol that we propose is based on the idea of exchanging *message objects* rather than *data segments* that are timed by the ACK mechanism of TCP. Message objects consist of a number of short sized (typically 64 or 128 bytes) data packets.

3.1. The case for CentaurusComm

Most of the prior work on improving TCP performance in wireless networks aims at reducing the negative effects of TCP's congestion control mechanisms. Protocols like WTCP implement a completely different congestion control mechanism in order to ensure that non-congestion losses are not handled in the same manner as congestion losses. Modifications to TCP, such as the addition of selective acknowledgments (SACK) [15], have also been proposed as solutions to the problems posed by TCP's congestion control mechanism in both wired and wireless networks. Ref. [5] dis-

cusses additions to TCP with SACK specifically tailored for wireless networks. However, all of these approaches, including those described in detail in the previous section, require TCP stack modifications on the mobile host, base station, wireless gateway and/or wired hosts that need to interact with the wireless devices. Such a requirement poses difficulties in deploying the solution across a wide range of networks and platforms. CentaurusComm, on the other hand, can be implemented on top of any wireless technology without any modifications to the underlying stack.

We believe that neither the wireless gateways which support the wireless devices nor the wireless devices themselves will experience enough congestion to justify the implementation of a heavyweight congestion control mechanism in our protocol. We note that the likelihood of congestion on the wireless device is negligible because it seldom establishes more than one link layer connection to the gateway or another wireless device. Now, consider the possibility of congestion due to inbound traffic (in the wireless device-to-wireless gateway direction). We contend that congestion cannot be caused in this direction for the following reasons. One obvious reason is the typically low bandwidth of the wireless hop as compared to the rest of the network. This ensures that even though the number of wireless devices that the wireless gateway may support is large, the devices cannot cause significant congestion. In addition, we note that wireless devices do not generate large enough amounts of data to cause buffers on the gateway to overflow. A third reason, specific to our protocol, is its use of message objects coupled with an acknowledgment mechanism, which ensures that no unnecessary retransmissions need to be performed, thus reducing the likelihood of congestion at the gateway. Finally, we believe that a majority of the wireless clients will engage in conversations of a short duration with the gateway, reducing the possibility of congestion. Next, we consider congestion on the gateway due to outbound traffic (in the wired nodes to wireless devices direction). The primary cause of congestion on the gateway in this direction is due to the fact that the wired nodes send data at higher rate then the effective bandwidth of the wireless link. Techniques like pacing [11] or the rate-based transmission control in WTCP slow down the rate from the wired node so that the wireless gateway does not become congested.

The ACK mechanism in TCP, while suitable for wired networks with high bandwidth and low delays, could cause unnecessary retransmissions leading to reduced throughput in wireless networks that typically have low bandwidth and high delay. As we show in Section 5.2.1, it is quite possible for the TCP ACKs from the receiver to be delayed in transit causing the sender to unnecessarily retransmit data, due to the retransmission timeout. This problem is very easily exacerbated by repeated retransmissions and duplicate ACKs leading to complete breakdown of transmission, eventually resulting in forced termination of the TCP session. TCP with SACK seeks to remedy this problem by allowing the receiver to inform the sender via ACKs, of up to 4 (in general 3) blocks of contiguous data separated by "holes" due to lost data segments. In [14], Allman et al. have analyzed TCP performance over satellite links, which experience long delays. TCP with SACK exhibits improved performance as compared to TCP Reno, over such links. However, in [6], Balakrishnan et al. have shown that TCP with SACK does not help improve performance if window sizes are small, which is the typical case of TCP implementations on wireless devices (e.g., in PalmOS). In addition, TCP with SACK suffers from the same congestion control related problems of standard TCP, if only one data segment is lost per window of transmission, as opposed to multiple segments. CentaurusComm, in contrast, uses a simple bitmap-based acknowledgment mechanism. This is similar to the SMART mechanism [12]. The main idea in SMART is for the sender to build a bitmask of correctly received packets at the sender, instead of carrying it in the ACK header. Thus, each ACK consists of two parts: the cumulative ACK, and the sequence number of the packet that caused the ACK to be initiated. This mechanism is further simplified in CentaurusComm. Both the sender and receiver maintain a bitmap, the former of packets transmitted and the latter of packets received. No signaling information is exchanged between the sender and receiver *during* data

transmission. Only after completing data transmission does the sender signal the receiver. The receiver checks its bitmap and determines whether or not all expected data packets have been received. We show in the following sections, how this method of checking is sufficient for the receiver to ensure that it has received all expected data packets. If some data packets have been lost, the receiver simply signals the sender via its own bitmap. The sender uses the received bitmap and its own, to determine which packets need to be retransmitted. If all packets were received, the receiver signals this fact via a different message. The main issue with this acknowledgment mechanism, we believe, is the possibility of increased timeouts at the sender, if either of the signaling messages are lost. However, these timeouts do not result in any unnecessary transmissions; only in increased session time.

Increasing the window size of TCP on both the wireless device and the gateway would seem to be another reasonable way of eliminating TCP problems in wireless networks. However, on small wireless devices like the Palm Pilot, memory is a scarce resource. It could be argued that the window size could be set to some small multiple of the segment size, say 2–4 segments. This could lead back to the original problem of TCP losses if larger data packets, say between 4 and 8 KB, are to be exchanged. For example, typical email sizes could range from a few hundreds of bytes for text only to about 8 KB when sent with attachments. The problem is that on small devices, the TCP segment size is limited to 256 or 512 bytes, resulting in a larger number of segments, say 8–16, being transferred. If a few of these are delayed in transit, they could set off the chain of events leading to invocation of congestion control by TCP, resulting in reduced throughput.

The idea of using a message based protocol over CDPD rather than TCP is also found in the Aether intelligent messaging (AIM) protocol [11]. Unfortunately, only a high level white paper that describes the protocol is publicly available, making it impossible for us to compare our approach with it. However, it does appear that AIM has an approach similar to CentaurusComm.

3.2. System architecture

Fig. 1 shows the design of the CentaurusComm architecture from the perspective of data exchanges. Fig. 2 shows the interaction of the components in the architecture based on the exchange of control messages. CentaurusComm consists of one or more lower level protocol modules (designated as Level I), one higher level module (designated as Level II) and an application program interface. Level I modules are communication medium dependent; the Level II module is medium

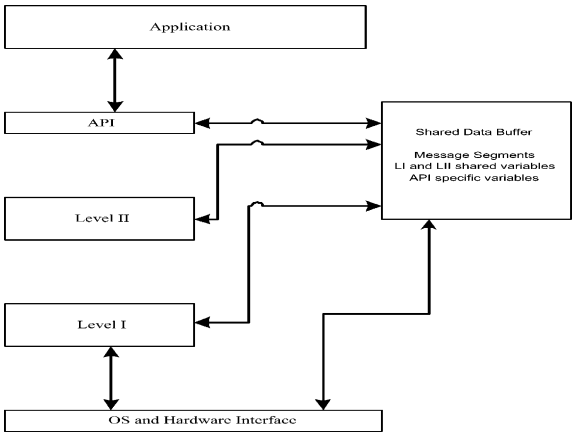

Fig. 1. Interaction between components of CentaurusComm protocol (data).

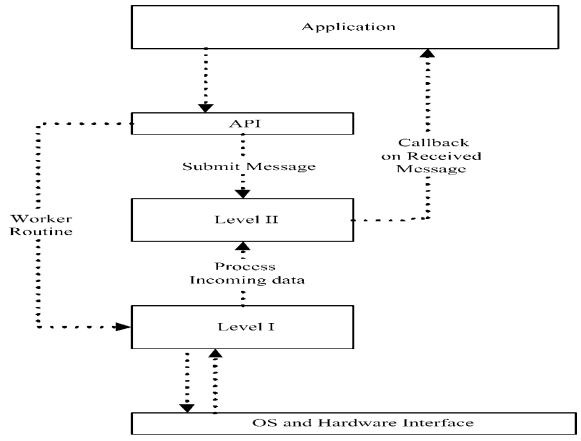

Fig. 2. Interaction between components of CentaurusComm protocol (control).

independent. The API is responsible for accepting the objects from the application layer for transmission and notifying it when messages are received.

The protocol is implemented as a collection of data structures and state machines. The principal data structures include the *transmit* queue and the *receive* queue. As the protocol is designed to run on a wide range of low power systems such as PDAs and low power embedded computers, it does not depend on any advanced operating system features such as signals and multithreading, that are typically not part of such systems. This is in contrast to TCP, which requires substantial support from the OS for signaling. CentaurusComm is designed such that the transmission, reception and recovery procedures are divided into many small sub-procedures that last for a very short time. With the exception of domain name resolution, which occurs very infrequently, the protocol never blocks. This design gives an impression of concurrent execution of the user program and the protocol modules.

The Level I and Level II protocol modules share data items in order to communicate with each other. The Level I module copies the contents of any received data packet (excluding the headers) to the common area and runs the Level II state machine. The Level II state machine examines the contents of the received packet, based on which, it changes state. If, based on the new state, a response is in order, the Level II module places the response in the common communication area. In addition, if the Level II state indicates that the session is finished, the Level II module sets a specific flag in the common area.

3.3. The Level I module

The principal driver of the CentaurusComm protocol is a *worker routine* that is part of the Level I module(s). On startup, the user application must always call this routine. The main purpose of the *worker routine* is to perform message transmission and handle message reception. Based on the status of the *transmit* and *receive* queues, the network connections and the Level II state machine, the worker routine takes one of the following actions:

- If data is present in the *transmit* queue, schedule it for transmission.
- If data is present in the *receive* queue, run the Level II state machine in order to process it.
- In peer-to-peer type networks, examine the table of outgoing messages and trigger the Level II state machine in order to start transmission.
- In IrDA-based networks, cause the "master" to periodically establish connections with the "slaves" and trigger the Level II state machine to start a session.

When the Level I module establishes the connection with the peer, it resets the state machine of the Level II protocol to the initial state. As long as the physical connection exists, every packet that is received by the Level I module is sent to the Level II protocol module.

Session setup and start up, as performed by Level I, are communication medium dependent. On media such as InfraRed, one of the devices is selected to be a master. This is the only device that can start a session. The master device is responsible for discovering all the devices in the neighborhood that it can communicate with and polling these devices for messages by establishing a session with each device in a Round-Robin fashion. For media that allow multiple nodes to communicate at the same time (either in point-to-multipoint or multipoint-to-multipoint mode) the device that has an outgoing message is responsible for establishing the session with the recipient. On such devices the Level I module is responsible for maintaining the state of sessions for different devices and loading the correct state for each session. Every time a packet is received from a particular device, the Level I module looks up the sender devices in its session table and if an entry exists for the device, it sets up the state machine of Level II module according to the entry in the session table. When the Level II module finishes processing the packet, the Level I module saves the new state back in the session table. This switching method is not the most efficient, but it provides for a reasonable argument against implementing a specialized Level II protocol module for each possible communication medium or having to use multithreading or multiprocessing features of the operating system (if they exist).

3.4. The Level II module

The Level II module performs reliable transmission of messages. It provides message segmentation and reassembly, keeps track of lost packets and performs retransmission using the acknowledgment mechanism described above. In addition, it provides some rudimentary time synchronization mechanisms along with identification and deletion of old messages. The current version of the Level II code works on PalmOS and Linux (both user space and kernel space).

3.4.1. Level II sessions

The Level II module consists of a session based protocol. Session management (setup, shutdown) is still the responsibility of the Level I module. In general, a single session may consist of a single message from each end point to the other. Thus, at most two messages can be transmitted in one session. Under certain conditions, a session may not be able to handle all the data packets sent by an end point. In such cases, the message may span multiple sessions. Multiple sessions may be required if the underlying communication medium does not allow more than two entities to communicate at the same time, thus requiring some type of time division multiplexing. InfraRed and Bluetooth are typical examples of such media. Multiple sessions per message may also be required if bad network conditions cause the loss of the control packet, which contains signaling information. For reasons of time and memory conservation, the CentaurusComm protocol does not provide any mechanism for retransmission of control packets. As is well-known, time and memory considerations are always important in the context of low power devices like PDAs. Therefore, when a packet that carries control information is lost, the session cannot continue and will hang till a watchdog timer destroys it. After the session is destroyed by the watchdog timer, a new session is created and the message transmission resumes. The acknowledgment mechanism ensures that data packets received in the previous session will not be retransmitted.

3.4.2. Transmission initiation

As described in Section 3.3, the Level II state machine is triggered by the Level I state machine. Depending on the type of network—IrDA-based or peer-to-peer—the Level II state machine is initialized to the *idle* or the *wait* state, respectively. (We use the abbreviation L2SM to denote the Level II state machine in the rest of this section.) The L2SM message sequence chart is shown in Fig. 3. We describe below, the message exchanges and corresponding state transitions (as far as the transmission initiation is concerned) for both IrDA-based and peer-to-peer networks.

Initial state transitions specific to IrDA-based networks. The L2SM changes to the *wait* state on all slave devices, after receiving a HELO message from the local Level I module. On the designated master device, once the Level I connection has been established with one or more slaves, a HELORESP message is sent to the L2SM. The L2SM on the master device then transitions to the *connected* state. This is done as an additional safeguard to prevent the master device from communicating with devices that do not support the CentaurusComm protocol. As the link latency in IrDA-based networks is quite low (around 500 ms), this safeguard does not cause unacceptable overhead. (However, the overhead in peer-to-peer networks is too high and does not justify use of the safeguard. Hence, the difference in initial state transitions of the L2SM.) In this state, it transmits a POLL message on the IrDA interface to determine if the slave devices have data to send, and transitions back to the *wait* state. When the L2SM on a slave device receives the POLL message, it transitions to the *poll received* state.

Initial state transition specific to peer-to-peer networks. The L2SM transitions from the *wait* state to the *poll received* state upon receiving a POLL message from the local Level I module.

The remaining state transitions, as described in this and the following sub-sections are common to both types of networks.

After transitioning to the *poll received* state, the Level II module scans the table of outgoing mes-

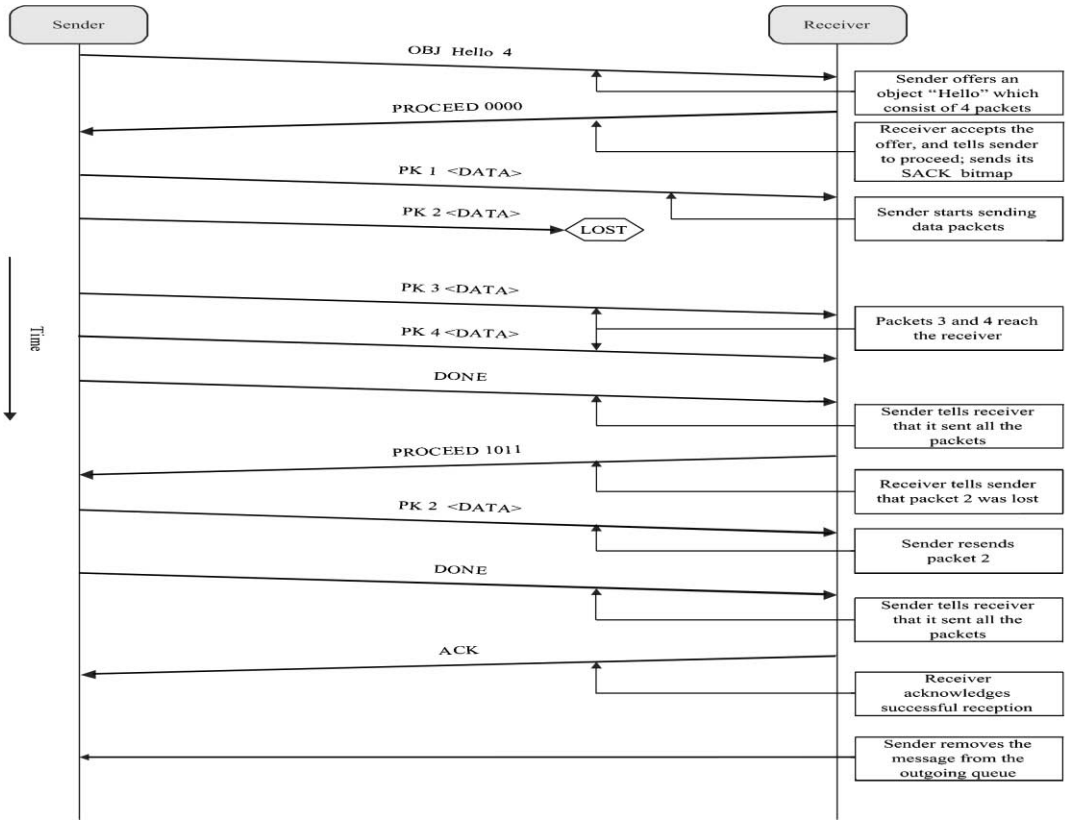

Fig. 3. Message exchange sequence for CentaurusComm protocol.

sage objects to find the one that should be delivered to its peer. On small mobile devices, this table typically contains only one slot which is either full or empty, so the selection of the object is trivial. On server class systems, this table will have multiple entries, therefore linear search is used to select the outgoing object. Once the object is selected, the Level II state machine sends an OBJ message (which contains the class of the message object, its size and the timestamp) to its peer (via the Level I module) and transitions to the *send command* state. As soon as the Level I module places the message on the network stack, it informs the Level II module via the SENT message. This forces the Level II state machine back into the *wait* state.

Upon reception of an OBJ message, the L2SM on the peer transitions to the *obj received* state. In this state, it examines the class and timestamp on the object and decides whether to accept or reject the object. Objects are rejected if either the class of the object is unacceptable or if the device already has a newer copy of the object of this class. If the device decides to reject this object it sends back a REJ message and transitions back to the *wait* state. Otherwise it sends back a PROCEED message that contains the bitmap, which indicates all the received data packets that are part of this message object. If this is a new message object, this bitmap has all its bits set to 0. In this case, the L2SM transitions to the *proceed sent* state. After the Level I has sent this message to the peer and confirmed the transmission, the L2SM transitions to the *packet receive* state.

3.4.3. Message object transmission

If the device that sent the OBJ message receives a REJ message in response, the L2SM transitions

to the *ack-rej received* state. In this state, it removes the object from the table of outgoing objects. In addition, it responds with a NOPE message and the L2SM transitions back to the *wait* state. On receiving the NOPE message, the L2SM on the peer transitions to the *nope received* state. In this state, it determines if it has any object to send. If so, it sends it the OBJ message; message exchanges and state transitions following this are exactly as described in the previous section. If this device has nothing to send to the peer, the session is closed and the L2SM returns to either the *idle state* or *wait* state.

If the device that sent the OBJ message receives a PROCEED message in response, the L2SM transitions to the *proceed received* state. In this state, the Level II module first updates its copy of the bitmap using the one present in the PROCEED message. The L2SM enters the *packet transmit* state. It then starts transmitting those data packets, in the message object, that correspond to the bitmap entries marked 0. The data packets consist of two fields—the packet header and the payload. The packet header consists of two fields—a two-byte constant corresponding to the string "PK" and a four-byte slot number. The process of packet transmission is as follows. After determining the correct data packet number to start sending from, the device prepares the packet by setting the destination slot number in the packet header followed by the payload. This packet is copied to the common communication area and a flag is set. This signals to the Level I module that data is ready for transmission. The L2SM transitions to the *wait for packet send* state. After the Level I module has transmitted the packet, it informs the Level II module which causes the L2SM to transition back to the *packet transmit* state, after updating the bitmap.

It then checks the bitmap and the object size, and continues to transmit packets as long as the bitmap contains any '0's. We note that the synchronous nature of session set up allows the L2SM to remain in the *packet transmit* state for the duration of transmission instead of switching between this and the *wait for packet send* states. If a control message is lost because of the problems with local network stack or during the transmission, the whole session will be terminated by the watchdog timer.

While L2SM on the source device is in the *packet transmit* state, the destination L2SM is in the *packet receive* state. The header of every data packet received is checked for the "PK" string. The payload is copied into the object receive buffer at the slot indicated in the packet header. The bit corresponding to this slot is set to 1 in the bitmap.

3.4.4. Transmission completion

After the source device completes sending all data packets in the message object, L2SM sends a DONE message to the peer and transitions back to the *wait* state. When the peer receives the DONE message, the L2SM transitions to the *done received* state. In this state, it checks its message bitmap; if all the expected data packets from the object have been received then it responds with the ACK message. It also updates the timestamp of the last received object in its table of accepted objects. It then sends an indication to the application that the message has been received. The application is responsible for processing the received message before it calls the worker routine again, because the contents of the message buffer might get overwritten on the subsequent run of the worker routine. The L2SM transitions back to the *wait* state. When the L2SM on the data source receives the ACK message, it transitions into the *ack-rej* state. If the data source is the master in IrDA-based networks or the "server" in peer-to-peer networks, it closes the session causing the physical connection to be terminated as well. If the data source is a slave or "client", then it sends a NOPE message to the other side and transitions back to the *wait* state. On receipt of a NOPE message, the master or server L2SM then behaves as described in Section 3.4.3.

If the destination L2SM receives the DONE message and determines via its bitmap that one or more segments of the message still have their corresponding bits set to 0, it sends a new PROCEED message and a new bitmap to the sender and transitions back to the *proceed sent* state.

4. Performance metrics and experimental setup

4.1. TCP performance metrics

- *Round trip time* (*RTT*): It is one of the most important measures of network performance. In our experiments, we have measured RTT on both the client and server. The client is primarily the initiator of connections, as described below. Thus, the client has the most accurate measure of RTT of packets on the channel. This is because the client starts recording time just before transmitting the request on the channel and stops recording immediately after receiving the reply.
  We record two approximations to the RTT on the server. These are approximations, because the server is not aware of the beginning of transmission from the client, nor is it aware of the time that the client received its reply. The first RTT includes the client request, server reply and a "good bye" message from the client. These three messages together constitute a session in all our experiments. The server tries to obtain a sense of the "real" RTT by starting to record time before replying and stopping recording after receiving the "good bye" message. This is the second approximation to the RTT.
- *Retransmitted TCP segments*: This metric provides an indication of how TCP is being affected by the current state of the network. Retransmissions on a wireless network may occur due to two reasons:
  - loss of signal between wireless client and base station,
  - congestion at some intermediate base station/wired node leading to packet drops.

  The TCP implementation in Linux 2.2.17 on our test server performs fast retransmits—retransmit only the segment not received by the client. Therefore, in the best case only one segment is retransmitted and in the worst case, the entire window is retransmitted. We record the number of retransmitted segments per request-reply-good bye session on both the server and the client.

  We attempt to obtain more information on retransmissions by recording the number of bytes and packets actually transmitted by the Ethernet driver on the server and the modem on the client. This information is more useful when obtained on the client than on the server because the latter may receive and transmit packets to other clients concurrently. We perform this experiment because we are able to analyze the effect of different message sizes on the number of retransmissions that TCP must perform. This could give us an idea of what the optimum size packet might be to transmit over the wireless channel. This information is recorded per session on both the server and client.

4.2. CentaurusComm performance metrics

- *Round trip time*: This parameter is measured on the client in a similar manner as described above. However, no RTT measurements are obtained on the server. Unlike TCP/IP, where the message has to traverse three layers before a timestamp can be applied to it, a message object has to traverse four layers. Thus, the accuracy of measurement is further reduced on the server. In order to obtain accurate RTT measurements on the server, we recorded and analyzed the output produced by the *tcpdump* program.

4.3. Measured CDPD parameters on client

- *Relative signal strength indication* (*RSSI*): RSSI is a parameter representing the received signal strength of both the wireless client and the base station. It is used by the client to determine whether a hand-off procedure or a power change must be initiated. This value is measured in dBm and on CDPD networks a value of -113 dBm indicates the absence of signal. This parameter very clearly indicates whether or not a client can receive and transmit data from its current location. We record the value of RSSI by querying the modem every second.
- *Forward block error rate* (*BLER*): This parameter measures the state of the channel from the perspective of noise and errors in transmission due to noise. This measurement is performed on the forward channel (base station to wireless client). The CDPD network uses the Reed-Solomon forward error correcting code (FEC) on

transmitted blocks of data. Each block consists of 420 bits including data, the FEC, and encryption information. Each block is split into 7 60-bit microblocks that are transmitted continuously. The BLER is calculated on the forward and reverse channels by the modem. The formula for calculating BLER is:

$$\mathrm{BLER} = (\mathrm{BS} - \mathrm{CBR})100/\mathrm{TBS} \quad (1)$$

where BS is blocks sent, CBR is correctable blocks received, TBS is total number of blocks sent.

A correctable block is one that had no errors or had errors correctable by the Reed–Solomon code. The Reed–Solomon code can correct errors in up to eight symbols per block. Each CDPD symbol consists of 6 bits. The reverse BLER is measured on the reverse channel (wireless client to base station).

- *Cell busy*: This is a measure of how busy the cell that supports the client is. This is also measured as a percentage. The client will not be able to transmit or receive data if the cell becomes too busy to support this client. Thus, the client can try to reach a base station in an adjoining cell that may not be busy.

4.4. General experimental setup

We used the typical client server scenario to perform experiments. The client programs were executed on an Intel Pentium running Linux 2.2.17 and Palm Pilots running PalmOS 3.5 using OmniSky modems for communication over CDPD. For evaluation of TCP and CentaurusComm performance on Bluetooth, we executed the client programs on an Intel Pentium running Linux 2.2.17 using the Bluetooth stack developed by Axis Communications, Inc. We used Bluetooth hardware developed by Ericsson. A concurrent server executed on an Intel Pentium running Linux 2.2.17. In all experiments, the wireless client initiates the connection to the server that is part of a LAN.

4.4.1. Application model

All experiments were done on a per-session basis. A session consisted of a request message from the client, a reply message from the server and a good bye message from the client. We have used the typical request/reply scenario found in most client/server based applications. We have attempted to model common applications like email (without attachments), Web browsers and FTP. Typical message sizes in these applications range from tens of bytes (e.g., emails) to thousands of bytes (e.g., WWW and FTP). In order to include these applications, we have therefore chosen our minimum and maximum message sizes as 64 and 8192 bytes respectively. In these applications, the client usually initiates the connection, sends a request and waits for server response. We performed different sets of experiments in which either the request message or the reply message size was constant. This was done in order to model the application scenarios more accurately (where the common case is for one side to send small nearly-fixed size messages and the other to send larger, variable size messages). The good bye message consisted of all recorded values on the client, of various parameters described above.

4.4.2. Palm pilot/cellular digital packet data specific setup

The experiments consisted of 100 sessions per variable packet size (128, 1024, 2048, 4096 and 8192 bytes). The following sets of experiments were performed.

- Variable request size: The client sends a request packet of size 128 bytes, the server replies with a constant sized packet (32 bytes in all experiments) and the client replies back with the good bye packet. This sequence is repeated a 100 times before the client request size is increased to 1024 bytes.
- Variable reply size: The client sends a constant request size of 32 bytes, the server replies with a 128 byte packet and the client replies back with the good bye packet. This sequence is repeated a 100 times before the server reply is increased to 1024 bytes.

Typically, the request, reply and good bye packets are sent and received after establishing a TCP connection between the client on the Palm

and the server on the Linux box. During experiments, we discovered that the limit on the number of open sockets on the Palm is 15. Thus, after the first 15 connections are established and sessions are in progress, attempts to establish more connections fail. It is also known that the connections remain in the TIME_WAIT state for a while even after the sockets have been closed. Thus, a large amount of time is spent in waiting to establish connections. In the original design of the experiments, 1000 sessions were to be established per variable packet size. However, due to the above problem, the overall testing time for 1000 sessions was well over 8 h. In order to reduce the testing time, we reduced the number of sessions to 100. In addition, we decided to establish only one TCP connection to exchange messages in all 100 sessions with packet sizes varying as described above. Thus, we eliminated the connection setup time from the overall testing time. In order to assure ourselves that connection setup time remains the same on the average, we included the connection setup and tear-down for 100 sessions with the client request size of 4096 bytes.

A side affect of the long testing time was complete discharging of the battery on the Palm Pilots even before one set of experiments was completed. Thus, with the above optimizations, all experiments were completed before the discharging of the battery.

Another observation we made during testing is that TCP packets of size 8192 bytes cannot be sent from Palm Pilots successfully. There seem to be some problems associated with sending large sized packets from the Palm. One problem is the small maximum segment size (MSS) of 536 bytes. A small MSS means that a large packet will be sent as many small fragments. We have analyzed the output of *tcpdump* on the server and it appears that the ACKs to all the fragments are not received by the Palm. Thus, the Palm assumes that the fragments were lost and retransmits one or more fragments. This leads to increased traffic and exacerbates the situation. The PalmOS networking function provides for a finite application timeout after which the application reports an error. We have observed that invariably, this timeout expires when packets of size 8192 bytes are sent. Increasing this application timeout made little difference to the transmission failure.

We note that CentaurusComm faced none of the problems discussed in the context of TCP. Large packet sizes posed no problems and were successfully transmitted. However, in order to limit testing time, only 100 sessions were created per variable packet size.

4.4.3. Bluetooth specific setup

As mentioned above, both the client and server programs were executed on Linux boxes, to test TCP and CentaurusComm performance over Bluetooth. Therefore, we did not face any of the Palm related problems discussed above. Experiments consisted of 1000 sessions with variable packet sizes of 64, 128, 1024, 2048, 4096 and 8192 bytes. The experiment sets were generated in the same manner as for CDPD. However, in order to exchange TCP/UDP packets over Bluetooth, the point-to-point protocol (PPP) must be pushed on top of the Bluetooth stack (consisting RFCOMM, L2CAP, serial transport driver and lower layers). PPP requires a connection to exist between the peers that need to communicate via TCP or UDP or any other transport protocol. Thus, an RFCOMM connection was first established between the two Linux boxes. PPP was then started and the connection remained established through all sets of experiments.

4.4.4. WLAN specific setup

Experiments over the WLAN were conducted using two laptops with similar CPU, memory and kernel configurations. Both laptops were running Linux 2.4.2. The laptops communicated with each other in infrastructure mode via a Cisco Aironet Base station, using SMC2632W 11 Mbps wireless PC cards. We noticed that the typical data rates achieved by the cards was around 4 Mbps. The client and server programs established 1000 sessions with variable packet sizes of 128, 256, 512, 1024, 2048, 4096 and 8192 bytes. In order to understand the behavior of both TCP and CentaurusComm over WLAN, we conducted two sets of experiments. In the first set, the two laptops exchanged data without any interference. In the second set of experiments, we set up two more

laptops with wireless PC cards in the infrastructure mode. We established a single TCP connection to port 19 (*chargen* service) in both directions. The *chargen* service continuously generates characters and transmits them on port 19.

4.4.5. Controlled experimental variables

In this section we define the overall experiment space and describe the various controlled variables. We have used five controlled variables in the experiments. These are packet size (request or reply), connection setup mode (on/off), time-of-day, location and client mobility. The last three variables are described briefly below. Our experimental space is defined by different combinations of these five variables. Based on the different values that these variables are allowed to take, we have quite a large experimental space consisting of about 288 (CDPD) to 432 (Bluetooth and WLAN) possible five-tuples. We allow each variable to take only a subset of the possible values and thus reduce the size of the space considerably.

- *Time of day*: This variable takes two values. One value indicates the peak hour periods during the day, the other value indicates non-peak hour periods. For peak hour periods, we chose a time period between 6:00 PM in the evening to 11:00 PM at night. For the non-peak hour periods, the time period chosen was 12:00 midnight to 6:00 AM.
- *Location*: Possible locations to perform the experiments include downtown, sub-urban, rural and small towns. Downtown areas would likely be more busy at most times during the day. Rural areas may almost never be fully busy. RSSI values could vary more rapidly in the downtown areas as opposed to rural areas. The physical placement and number of base stations in each type of area would affect the performance of TCP on both the client and server.
- *Mobility*: The wireless client in our experiments may be static, locally mobile (within a cell) or mobile between cells. Experiments done on a static client would show less variability of RSSI, and BLER. Experiments done when the client is mobile within a cell would show the effects of RSSI based on only one or two base stations. Inter-cell mobility experiments could show large variability of the uncontrolled variables.

5. Experimental results and discussion

5.1. TCP over Ethernet

In order to compare the performance of TCP over wired networks to TCP over wireless networks, two Linux boxes were connected across a wired network. One workstation was located on campus and had 100 BaseT connection to the campus wide network. The other workstation was located at the home of one the authors and was connected to @Home network via 10 BaseT interface. Packet sizes ranged from 64 to 8192 bytes, as described in the general experimental setup. The graph in Fig. 4 shows typical TCP performance over a high speed Ethernet link. The graph clearly shows that for small message sizes the performance of the protocol is extremely good. We can also infer from the graph that message size is not related to the transmission time.

5.2. Performance of TCP and CentaurusComm over CDPD

5.2.1. TCP over CDPD

Three different locations were chosen to measure performance of TCP over CDPD. Each location observed different signal strengths, peak hours and channel busyness. However, our anal-

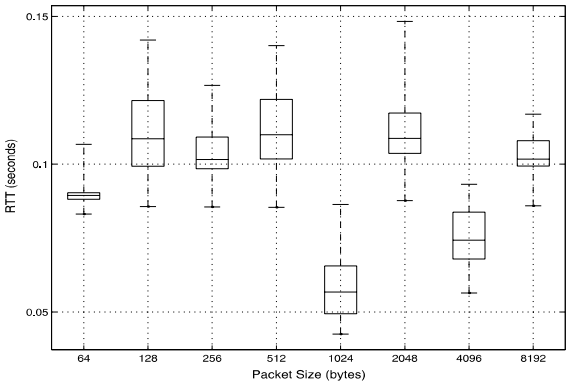

Fig. 4. Performance of TCP over a wired network.

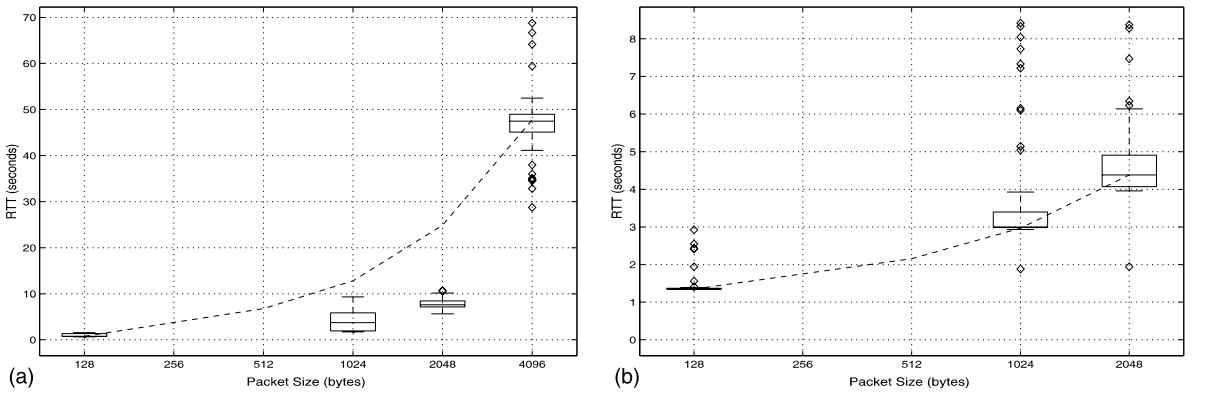

Fig. 5. Performance of TCP over CDPD: (a) static client, (b) mobile client.

ysis indicates that the overall performance of a completed transmission was not very different. The graph in Fig. 5(a) shows typical TCP performance over CDPD with respect to RTT. We were unable to obtain enough useful data for packets of sizes 256 and 512 bytes. However, we show the expected plot (dashed curve) if the RTT were to vary linearly with the packet size. It should be noted that the plot is a curve and not a straight line because the scale on the x-axis is logarithmic to base 2. We note that there is a sudden increase in RTT when the packet size increases from 2048 to 4096 bytes. Analysis of the output of *tcpdump* indicates that a large number of retransmissions begin to occur when transmitting segments of a 4096 byte packet. These retransmissions cause further segment losses and thus increase the overall time to transmit the entire packet, and in some cases it causes the interruption of the whole transmission process. This is a well-known problem; a good description of this problem can be found in [20]. An experiment in which the wireless client was mobile, was also performed. The speed was about 60 MPH. The graph for this experiment is shown in Fig. 5(b).

The graphs [4] in Figs. 6–8 show influence of the measured environmental factors on the performance of TCP for specific packet sizes. In Figs. 6 and 7 the TCP packet size per session is 1024 bytes and in Fig. 8 it is 4096 bytes. The plots of the

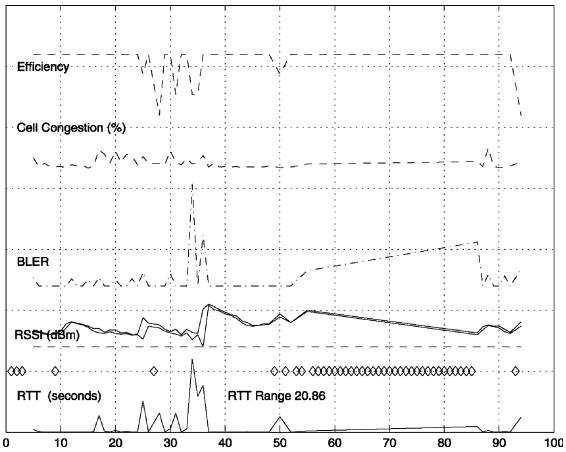

Fig. 6. Influence of environmental factors on TCP performance (mobile client, packet size = 1 K).

measured variables have been scaled and shifted on each graph. This allows us to visually determine the effect of RSSI, BLER and cell congestion on RTT. The diamond shapes on the graphs represent sessions that were dropped. The string of diamonds seen in the graph for the 'mobile' experiment represents going through an underwater tunnel. "Efficiency" is simply the ratio of number of bytes transmitted to those received per session. Ideally efficiency should be 1. When more number of bytes are received for a given packet size, due to retransmissions, efficiency decreases. The "RTT Range" shown in these figures is the difference between maximum and minimum RTT values for

[4] In these graphs no units have been specified on the y-axis due to multiple plots per graph.

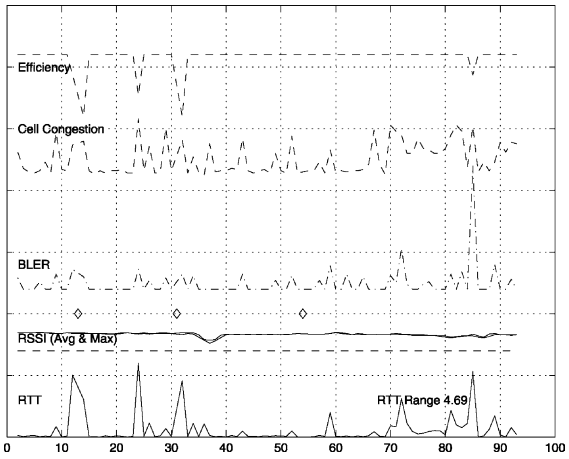

Fig. 7. Influence of environmental factors on TCP performance (downtown area, packet size = 1 K).

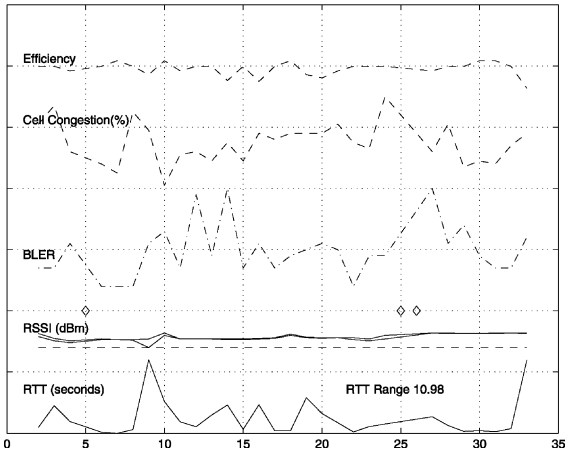

Fig. 8. Larger TCP transmissions are more prone to performance loss.

the chosen number of data points. We have not indicated any units on the y-axis in these graphs because the scale of each plot is different. The x-axis shows the packet number.

These graphs suggest that signal strength does not have much influence on the performance except when it drops to the zero level. The zero level is represented in the graph by the dashed line directly underneath the RSSI plot. The zero level for a CDPD modem is -113 dBm. We note that block error rate caused by RF noise and congestion at the cell cause significant performance degradation. In Fig. 6, we can clearly see that both efficiency and RTT are adversely affected by an increase in BLER. In Fig. 7, we see that a combination of moderate BLER and cell congestion lead to higher RTT and lower efficiency values.

Unfortunately due to the nature of the TCP client we were not able to record instances when modem is switching cells or gets ejected from the channel by mobile voice calls. The TCP client uses the native TCP stack on PalmOS for transmission. The source code of this stack is proprietary, so we were unable to keep track of the modem status and maintain TCP connection at the same time to determine when it would switch cells or possibly be ejected from a channel. When either cell switching or channel ejection occurs the transmission process is often aborted.

5.2.2. CentaurusComm over CDPD

Experimental results indicate that the CentaurusComm protocol is not as prone to performance degradation as TCP over CDPD. The graph in Fig. 9 shows CentaurusComm performance for different message sizes. When comparing the RTT values for CentaurusComm to those of TCP, it must be noted that TCP measurements do not include time required to set up a connection. The typical connection time for TCP under favorable conditions is about 2 s. So taking the connection time into account, it could be observed that for small packet sizes the performance of CentaurusComm is comparable to the that of TCP. For

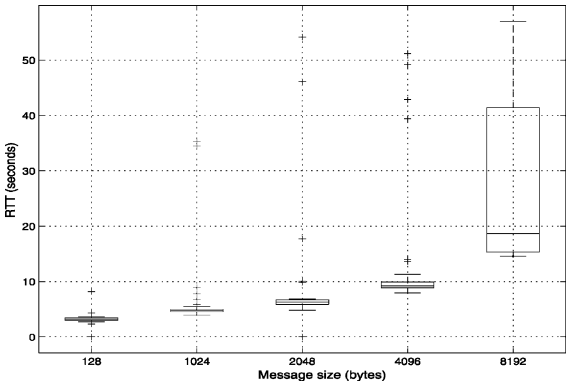

Fig. 9. Performance of CentaurusComm over CDPD.

message sizes of 4096 bytes the performance is better by a factor of about 5. For a message size of 8192 bytes measurements were not possible on TCP due to very few successful sessions. On the other hand, with CentaurusComm there were no lost sessions observed for any message size. However we did not test CentaurusComm under extreme conditions such as underwater tunnels etc. The graphs [4] in Figs. 10 and 11 show the relationship between environmental conditions and RTT. The packet size in Fig. 10 is 128 and 8192 bytes in Fig. 11. These figures also bring out the effects of BLER, RSSI and Cell Congestion on RTT. Most of the large peaks on the RTT plot were actually caused by the CDPD modem getting ejected from the channel and performing wide channel scan. We also found that CentaurusComm incurs a constant performance penalty of about 2 s because of the initial and final synchronization of the sender and receiver. However, this could be improved by redesigning the protocol so that it starts sending the beginning of the message before it gets the PROCEED message from the receiver. This will greatly improve the performance in cases when the environmental conditions are favorable, and it will not significantly reduce the performance under unfavorable conditions.

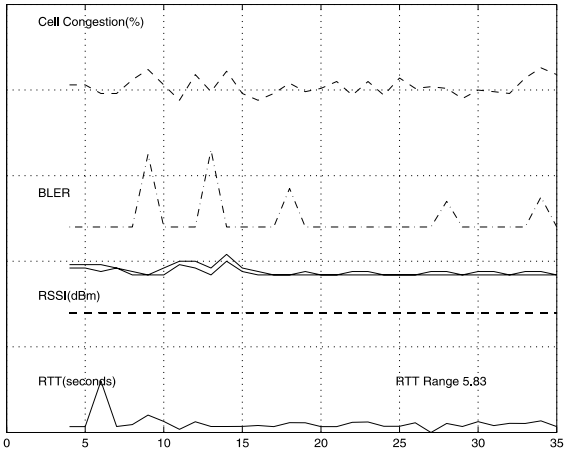

Fig. 10. Influence of environmental factors on performance of CentaurusComm in CDPD for 1 KB transmissions.

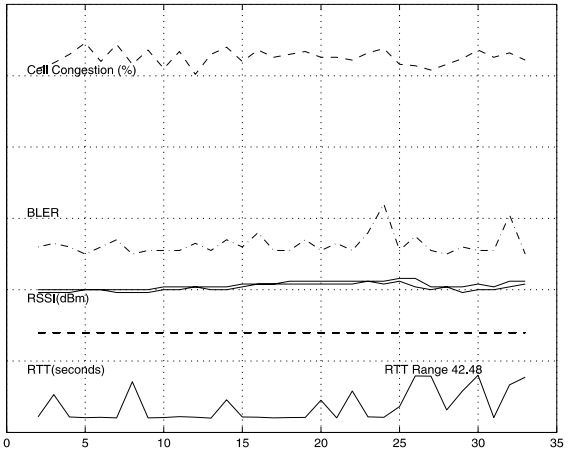

Fig. 11. Influence of environmental factors on performance of CentaurusComm in CDPD for 4 KB transmissions.

5.3. Performance of TCP and CentaurusComm over Bluetooth

5.3.1. TCP over Bluetooth

Analysis of TCP performance over Bluetooth indicates that the RTT remains almost constant for packet sizes between 64 and 1024 bytes, and again between 2048 and 4096 bytes. We notice a jump of around 400 ms in the RTT when the packet size increases from 1024 to 2048 and again from 4096 to 8192 bytes. It is possible that this behavior is due optimization of the TCP buffer size for certain sizes like 1024 and 4096 bytes. However, we have not investigated this possibility. The graph shown on Fig. 12(a) is typical of TCP performance over Bluetooth under the conditions described above. The analysis of the session dump shows that due to the low latency of the Bluetooth network, TCP does not exhibit the congestion control problem described in [20].

However, we did not perform extensive testing under different conditions, so it is not known how Bluetooth networks will behave under hostile conditions, such as high RF interference and mobility of the client. The testing performed Bluetooth testing on Unix workstations, which can afford to maintain large buffers and windows for handling transmissions. The behavior of TCP over Bluetooth might be different for small handhelds and embedded devices which are viewed as major market for Bluetooth. Testing TCP over Bluetooth using smaller devices is part of our future work.

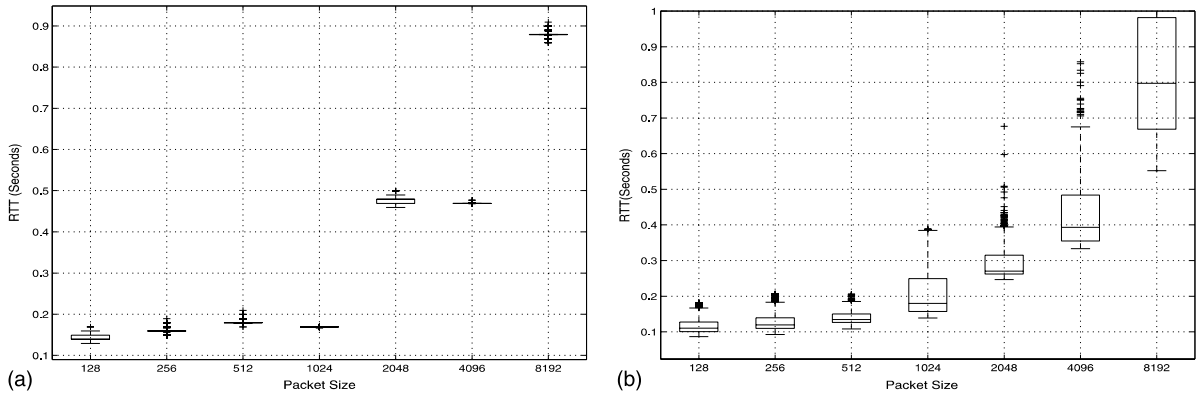

Fig. 12. Performance over Bluetooth: (a) TCP, (b) CentaurusComm.

5.3.2. CentaurusComm over Bluetooth

We conducted some preliminary experiments of CentaurusComm behavior over Bluetooth networks. The same Level I module that was used for running CentaurusComm over CDPD type of networks was used in this case as well. Analysis of CentaurusComm performance over Bluetooth shows that the protocol scales in linearly with message size. When comparing TCP and CentaurusComm over Bluetooth it should be noted that TCP shows better performance because it has the advantage of running in kernel space. In general, we find that CentaurusComm performs very similarly to TCP for small packet sizes (difference in RTT is around 50 ms up to 512-byte packets). For packet sizes 1024 bytes and above, the difference in RTT ranges between 100 and 150 ms. We attribute this to the slower execution of segmentation and reassembly in CentaurusComm (user space execution). One peculiar problem we faced with CentaurusComm was with 8192 byte sized packets. For reasons yet undetermined, the average RTT for 8192 byte packets increases phenomenally to around 3.5 s (from an average of around 0.6 s for 4096 byte packets). We are trying to determine the cause(s) of this behavior. The graph in Fig. 12(b) shows the performance of CentaurusComm over Bluetooth for different message sizes up to 8192 bytes.

5.4. Performance of TCP and CentaurusComm over WLAN

5.4.1. TCP over WLAN

Fig. 13(a) shows the performance of TCP over WLAN under normal conditions. We can see that

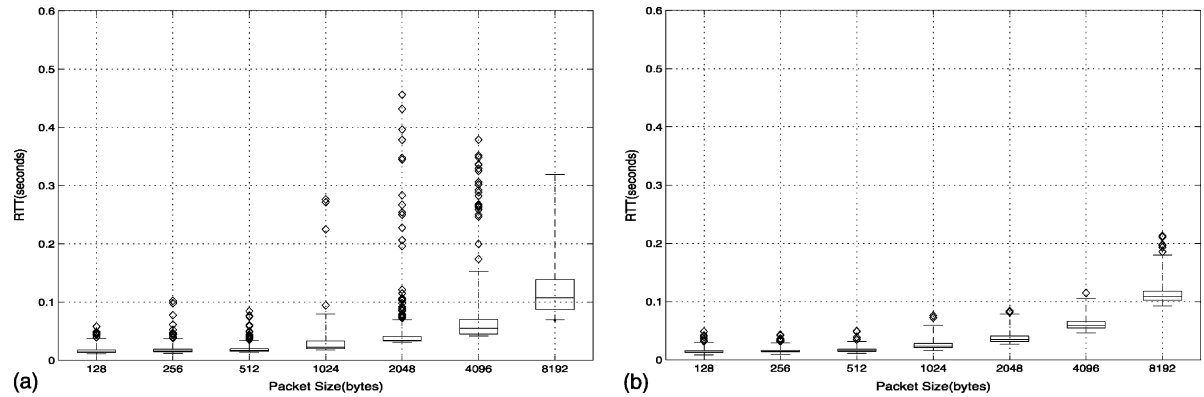

Fig. 13. Performance of TCP over WLAN under normal conditions: (a) Window Size = 2 K, (b) Window Size = 32 K.

TCP shows excellent performance over WLAN, almost as good as that over the wired network. This reinforces that well-known idea that TCP performs very well over high bandwidth networks. WLANs now provide bandwidth comparable to wired LANs and in typical settings are not the bottleneck. We do notice that a small percentage of sessions for each packet size show higher than average RTT values. We attribute this to normal congestion related issues at the access point that serves wired/wireless hosts and TCP's response to congestion. In Fig. 14, we show a combined graph of TCP and CentaurusComm performance over WLAN under adverse conditions as explained in Section 4.4.4. We have performed the experiment using only the 8192 byte packet size. The reason for this is that we observed that small sized packets were not significantly affected by the generated interference. The boxplot in the center refers to unmodified TCP performance under interference. Compared to the performance of TCP over WLAN under normal conditions (only 8192 byte packets), we note a degradation of around 400%. We explain the other two plots in Section 5.4.2.

5.4.2. CentaurusComm over WLAN

The performance of CentaurusComm over WLAN is shown in Fig. 15. Comparing this graph to the TCP performance graph in Fig. 13(a), we observe that the latter consistently outperforms CentaurusComm. For all packet sizes, the increase

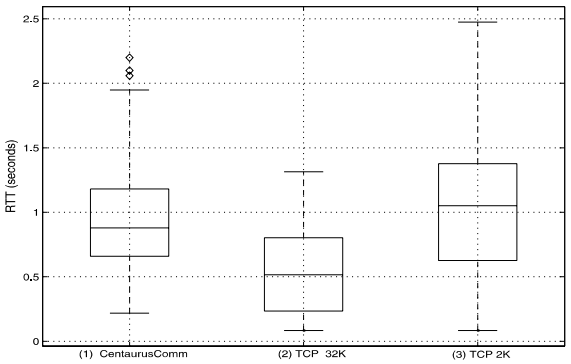

Fig. 14. Performance of TCP and CentaurusComm over WLAN under adverse conditions.

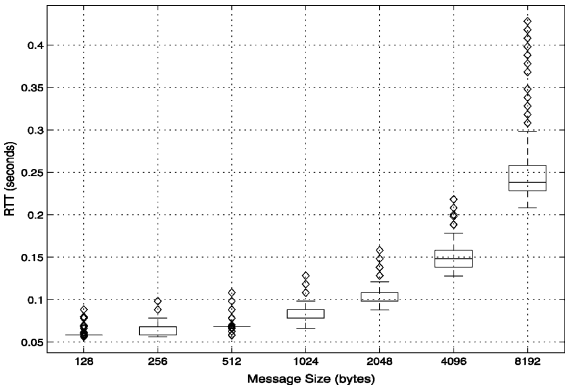

Fig. 15. Performance of CentaurusComm over WLAN under normal conditions.

in RTT for CentaurusComm is around 50%, compared to TCP. One reason for this performance differential could be the fact that the default window size used by TCP (in Unix kernels) is 32 K. This allows TCP to buffer as many segments of a particular packet as required, irrespective of its size. This, in turn, implies that once all segments have been received, the ACK can be sent back immediately. We have already noted that CentaurusComm has been designed for resource poor devices, whose memory capability is quite small. Thus, typical window sizes on these devices are around 1–2 K bytes. Thus, segmentation by CentaurusComm results in segments 64 bytes in length. As described in Section 3, the final ACK from the receiver to the sender is not sent until the entire message is received. This delay leads to an overall increase in the RTT. In order to attempt to validate this line of reasoning, we reduced the window size of TCP to 2048 bytes and performed the experiments again. Fig. 13(b) shows this graph. We notice a slight increase in RTT, leading us to believe that further reduction would result in RTT values comparable to those shown by CentaurusComm.

The experiments with interference were performed on CentaurusComm as described in Section 4.4.4. The plot in the first column in Fig. 14 shows the performance of CentaurusComm under adverse conditions. The second column shows the performance of TCP with the standard window size of 32 KB. The third column shows the

performance of TCP with the window size reduced to 2048 bytes. We observe that CentaurusComm performs better than TCP with 2048 byte window size, although TCP with 32 KB window size performs the best.

6. Conclusions and future work

In this work we have conducted an empirical study of the performance of two different transport protocols over wireless networks. We have shown that TCP, which performs well on wired networks, does not work as well in the constrained world of low bandwidth wireless networks. This confirms the results obtained from simulations done in other previous work. The main contributions of our work are

- to show, using empirical results, that TCP performs very poorly in low bandwidth, high latency wireless networks like CDPD
- to evaluate the performance of CentaurusComm, a protocol optimized for wireless networks

Finally, we also conclude that results obtained empirically are very important in evaluating performance of protocols in wireless networks. In addition we conclude that experimental studies bring out the effects of environmental conditions in wireless networks on various protocols more clearly than simulation studies. In the future we plan to conduct more rigorous testing of these protocols on Bluetooth and other types of wireless networks. At the time of writing of this paper, Level I modules for IrDA on PalmOS and Linux (kernel module), UDP on PalmOS and Linux (user space module) have been implemented. Level II modules have been implemented for PalmOS and Linux. The versions for Bluetooth on PalmOS and Linux are under development.

Acknowledgement

This work was supported by NSF awards IIS 9875433 and CCR 0070802.

References

[1] NS Simulator, University of Berkeley, California, http://www-mash.cs.berkeley.edu/ns/ns.html/.
[2] J. Ahn, P. Danzig, Z. Liu, L. Yan, Evaluation of TCP Vegas: Emulation and experiment, in: Proceedings of ACM SIGCOMM, 1995.
[3] A. Barke, B.R. Badrinath, I-TCP: Indirect TCP for mobile hosts, in: Proceedings of ICDCS, 1995.
[4] B. Bakshi, P. Krishna, N. Vaidya, D. Pradhan, Improving performance of TCP over wireless networks, in: Proceedings of ICDCS, 1997.
[5] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE ACM Transactions on Networking 5 (6) (1997) 756–769.
[6] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, M. Stemm, R.H. Katz, TCP behavior of a busy internet server: Analysis and improvements, in: INFOCOM, 1998.
[7] K. Brown, S. Singh, A network architecture for mobile computing, in: Proceedings of INFOCOM, 1996.
[8] R. Cáceres, L. Iftode, Improving the performance of reliable transport protocols in mobile computing environments, IEEE Journal on Selected Areas in Communications 13 (5) (1995) 850–857.
[9] K. Fall, S. Floyd, Simulation-based comparisons of Tahoe, Reno and SACK TCP, Computer Communication Review 26 (3) (1996) 5–21.
[10] C. Fang, H. Chen, J. Hutchins, A simulated study of TCP performance in ATM networks, ATM Forum Contribution 94-0119, 1994.
[11] AetherSystems Inc., Aether Intelligent Messaging.
[12] S. Keshav, S.P. Morgan, SMART retransmission: Performance with overload and random losses, in: INFOCOM, 1997.
[13] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, A. Joseph, Multi-layer tracing of TCP over a reliable wireless link, in: Proceedings of ACM BIOMETRICS, 1999.
[14] A. Mark, C. Hayes, H. Kruse, S. Ostermann, TCP performance over satellite links, in: 5th International Conference on Telecommunication Systems, 1997.
[15] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgment Options, http://www.rfc-editor.org/rfc/rfc2018.txt, October 1996.
[16] J. Mo, R. La, V. Anantharam, J. Walrand, Analysis and comparison of TCP Reno and Vegas, in: Proceedings of IEEE INFOCOM, 1999.
[17] J. Postel, Transmission Control Protocol, Network Information Center RFC 793, 1981, pp. 1–85.
[18] B. Rathke, M. Schlager, A. Wolisz, Systematic Measurement of TCP Performance over Wireless LANs, TKN Technical Reports Series TKN-01BR98, 1998.
[19] K. Ratnam, I. Matta, WTCP: An efficient mechanism for improving TCP performance over wireless links, in: Proceedings of Third IEEE Symposium on Computer and Communications (ISCC'98), 1998.

[20] P. Sinha, N. Venkitaraman, R. Sivakumar, V. Bharghavan, WTCP: A reliable transport protocol for wireless wide-area networks in: ACM MOBICOM'99, 1999.
[21] T. Goff, J. Moronski, D.S. Phatak, V. Gupta, Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments, in: Proceedings of the IEEE INFOCOM'2000, Tel-Aviv, Israel, vol. 3, 2000, pp. 1537–1545.
[22] M. Taylor, W. Waung, M. Banan, Internetwork Mobility: The CDPD Approach, Prentice-Hall, Englewood Cliffs, NJ, 1996.

Sasikanth Avancha is a Ph.D. student of Computer Science in the Department of Computer Science and Electrical Engineering at the University of Maryland Baltimore County. His research interests include wireless networks, mobile computing, and distributed systems. He obtained his M.S. degree in Computer Science from UMBC and his Bachelor of Engineering degree in Computer Science and Engineering from Bangalore University, India. He has over 5 years of experience in network software development and maintenance.

Vladimir Korolev is a graduate student in the Department of Computer Science and Electrical Engineering at the University of Maryland Baltimore County. He has got a Bachelor of Science degree in Computer Science from the University of Maryland Baltimore County. His research interests are in the fields of mobile computing and computer graphics.

Anupam Joshi is an Associate Professor of Computer Science and Electrical Engineering at UMBC. Earlier, he was an Assistant Professor in the CECS Department at the University of Missouri, Columbia. He obtained a B.Tech degree in Electrical Engineering from IIT Delhi in 1989, and a Masters and Ph.D. in Computer Science from Purdue University in 1991 and 1993 respectively. His research interests are in the broad area of networked computing and intelligent systems. His primary focus has been on data management for mobile systems in general, and most recently on data management in mobile ad hoc networks. He has created intelligent agent based middleware to support mobile access to networked computing and multimedia information resources. He is also interested in Data/Web Mining and Semantic Web, where he has worked on applying soft computing techniques to personalize the web space. His other interests include content-based retrieval of video data from networked repositories, and networked HPCC. He has published over 50 technical papers, and has obtained research support from NSF, NASA, DARPA, DoD, IBM, AetherSystens, HP, AT&T and Intel. He has presented tutorials in conferences, served as guest editor for special issues for IEEE Personal Comm., Comm. ACM etc., and serves as an Associate Editor of IEEE Transactions of Fuzzy Systems. At UMBC, Joshi teaches courses in Operating Systems, Mobile Computing, and Web Mining. He is a member of IEEE, IEEE-CS and ACM.

Timothy Finin is a Professor in the Department of Computer Science and Electrical Engineering and director of the Institute for Global Electronic Commerce at the University of Maryland Baltimore County (UMBC). He has over 30 years of experience in the applications of AI to problems in information systems, intelligent interfaces and robotics. He holds degrees from MIT and the University of Illinois. Prior to joining the UMBC, he held positions at Unisys, the University of Pennsylvania, and the MIT AI Laboratory. Finin is the author of over 140 refereed publications and has received research grants and contracts from a variety of sources. He has been the past program chair or general chair of several major conferences, is a former AAAI councilor and is AAAI's representative on the board of directors of the Computing Research Association.

Yelena Yesha received the B.Sc. degree in Computer Science from York University, Toronto, Canada in 1984, and the M.Sc. and Ph.D. degrees in Computer and Information Science from The Ohio State University in 1986 and 1989, respectively. Since 1989 she has been with the Department of Computer Science and Electrical Engineering at the University of Maryland Baltimore County, where she is presently a Verizon Professor. In addition, from December 1994 to August 1999 Dr. Yesha served as the Director of the Center of Excellence in Space Data and Information Sciences at NASA. Her research interests are in the areas of distributed databases, distributed systems, mobile computing, digital libraries, electronic commerce, and trusted information systems. She published 8 books and over 100 refereed articles in these areas. Dr. Yesha was a program chair and general co-chair of the ACM International Conference on Information and Knowledge Management and a member of the program committees of many prestigious conferences. She is a member of the editorial board of the Very Large Databases Journal, and the IEEE Transaction on Knowledge and Data Engineering, and is editor-in-chief of the International Journal of Digital Libraries. During 1994, Dr. Yesha was the Director of the Center for Applied Information Technology at the National Institute of Standards and Technology. Dr. Yesha is a senior member of IEEE, and a member of the ACM.