Transport Protocols in Wireless Networks

S. Avancha, V. Korolev, A. Joshi, T. Finin
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
Email: {savanc1, vkorol1, joshi, finin}@csee.umbc.edu

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 (CDPD) and Bluetooth. This analysis brings out the weaknesses of TCP in realistic conditions. 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 show that TCP performance on CDPD is very poor because of its low bandwidth and high latency. CentaurusComm outperforms TCP on CDPD. We show that on Bluetooth, which has higher bandwidth and lower latency than CDPD, both protocols perform comparably.**

I. 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 WLANs [2] and WANs based on packet radio technologies like CDPD [3] and General Packet Radio Service (GPRS) [4]. 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 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 ([5], [6], [7], [8]). 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 [9]. If the losses are not due to congestion, then TCP unnecessarily reduces throughput leading to poor performance.

This work is supported in part by NSF awards 9875433 and 0070802.

We propose a new protocol called CentaurusComm that eliminates the congestion related problems of TCP, while providing 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 show that this protocol performs better than TCP in CDPD networks and comparably to TCP in Bluetooth. In Section III, 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 [10]. 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. In simulators, these parameters must be carefully modeled in order to create realistic scenarios. 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 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 performance in wireless networks.

We now present a brief overview of the wireless network technologies considered in this paper – CDPD and Bluetooth. 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 19.2Kbps. 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). 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 rest of the paper is organized as follows: Section II discusses some well-known solutions for improving TCP performance over wireless networks and also describes prior work on empirical performance evaluation of TCP. We describe the CentaurusComm protocol in detail in section III. Section IV describes the performance metrics for TCP and CentaurusComm, the CDPD parameters we have considered and the experimental We discuss and analyze the experimental results in Section V. Section VI presents our conclusions and describes future work.

II. 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., I-TCP) or by replacing it by 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 violation of end-to-end semantics([11], [12]), modification of TCP code on the mobile client, the wired source or both ([13],[14]) or introduction of TCP-aware smarts in the base station [15]. Other solutions that attempt to improve TCP performance while retaining the end-to-end semantics include [16] and [17]. We describe, in brief, protocols discussed in [11], [12] and [17].

Indirect-TCP (I-TCP) was introduced in [11]; 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 a wired host is unaware of non-congestion related losses on the wireless part of the connection. A protocol similar to I-TCP, called MTCP was proposed in [12]. 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 solution proposed by [17] 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 [17] 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 were carried out, but the authors warn against using the results to quantify TCP performance because the experiments were performed in uncontrolled conditions.

Empirical TCP performance evaluation over some wireless networks is discussed in [15], [18] and [19]. In [19], throughput was chosen as the performance metric; 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 [18], interaction between TCP and Radio Link Protocol (RLP) (a GSM network specific link layer protocol) was the basis of performance analysis. The primary performance metric was TCP's utilization of the bandwidth provided by RLP. This work concludes that link layer solutions can alone solve the problem of poor TCP performance in wireless networks. Various solutions discussed above were actually implemented and TCP performance was evaluated over a WLAN in [15]. This work also concludes that a reliable, TCP-aware link layer provides very good performance.

III. CentaurusComm Transport Protocol

We now discuss a solution that seeks to replace TCP 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 and E-mail clients on their devices. These applications generate short messages as opposed to continuous streams of data. Our protocol 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 packets that are timed by the ACK mechanism of TCP. Message objects consist of a number of short sized data packets along with a bitmap. The basic idea is for the recipient of the message to only reply with an ACK after having received all data packets in the object. The recipient would use the SACK mechanism to indicate any data packets not received. We claim that this mechanism of data exchange is better than, for example, simply increasing the window size of TCP or using TCP with SACKs. Increasing the TCP window size would mean accommodating more data segments per transmission. Using SACKs with TCP implies that if data segments within the sequence are lost, those that have been successfully received would have to be buffered while waiting for retransmissions of the lost segments. In both cases, more memory and CPU cycles have to be

spent by the receiver. Typical clients in wireless environments cannot afford the extra overhead. We therefore believe that our solution is much better suited for resource poor clients in low to moderate bandwidth networks than TCP. It is, of course, possible to further modify TCP to make it more adaptable to wireless networks. However, we believe that with all the modifications included, the semantics of TCP would have changed so significantly that it would not resemble the original protocol design and thereby remain TCP in name only.

The idea of using a message based protocol over CDPD rather than TCP is also found in the Aether Intelligent Messaging (AIM) protocol [20]. Unfortunately, this protocol is proprietary to Aether Systems Inc. No specifications or implementations are publicly available. The information on AIM is provided only in a high level white paper. Thus, we were unable to compare the performance of our approach with that of AIM. However, it does appear that AIM has an approach similar to CentaurusComm.

A. System Architecture

CentaurusComm consists of two protocol modules (Level I and Level II) and an application program interface (API). Level I modules are communication medium dependent; the Level II module is medium independent. The API is responsible for accepting the objects from the application layer for transmission and notifying it when messages are received.

The entire protocol is implemented as a collection of data structures and state machines. 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. In fact, TCP requires signals in order to determine if certain conditions have occurred The protocol is designed in such a way that every step is performed in small chunks, with each chunk lasting for a very short time. With the exception of domain name resolution that occurs very infrequently, the protocol never blocks. The application is only responsible for calling a worker routine of the protocol.

The worker routine is part of the Level I module. Its main purpose is to perform message transmission or reception depending on transmit and receive queues. This routine checks the *send* and *receive* queues, the network connections and the Level II state machine. If required, the worker routine will attempt to send any data waiting in the *send* queue and/or run the Level II state machine to act upon data received from the *receive* queue. In addition, on peer-to-peer type networks, the worker routine will examine the table of outgoing messages and trigger the Level II state machine in order to start the transmission of outgoing messages, if this event has not yet occurred. On master-slave type networks, this routine is responsible for periodically establishing connections with the slaves and triggering the Level II module to start a session.

B. Communication between Level I and Level II modules

Both Level I and Level II modules are implemented as state machines. Level I and Level II share data items that serve as the communication area between the two modules. The Level II state machine is always set in motion by the Level I module. On master-slave type networks (e.g., IrDA), this is done if the Level I module is in the *connected* state. On peer-to-peer type networks (e.g., UDP/IP) this is done when a data packet destined for the Level II module arrives at the Level I module. In both cases, the Level I module will copy the contents of the packet (minus the headers) to a common area before starting the Level II state machine. The Level II state machine will examine the contents of a received packet and change its internal state as required. In addition, it places outgoing data (to the other end) in the common area. The Level II module sets a specific flag in the common area when a session ends.

C. 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 SACK mechanism. In addition, it provides rudimentary time synchronization mechanisms along with identification and deletion of old messages.

C.1 Level II Sessions

The Level II module consists of a session based protocol. During a session, both ends attempt to transmit a single message to each other. Thus, at most two messages can be transmitted in one session. However, under certain conditions, more than one session is required to transmit a message. Multiple sessions may be required if the underlying communication medium does not allow more then 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 may also be required if network conditions cause loss of a control packet. In order to conserve time and memory, the CentaurusComm protocol does not provide any mechanism for retransmission of a control packet. Therefore, when a packet that carries a control message 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. Now, because of the way SACKs are implemented, message data that was received in the previous session will not be retransmitted. The Level II module is not responsible for setting up and shutting down of the sessions. All session management is provided by the Level I module.

C.2 Session and Transmission Setup

When the Level I module establishes a connection with its peer, it resets the Level II state machine to the initial state and goes into the *connected* state. In the *connected* state, every

packet that is received by the Level I module is sent to the Level II module. Session startup is different for different types of communication media. 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 it can communicate with and polling these devices for messages by establishing a session with each one in a round-robin fashion. For media that allow multiple nodes to communicate at the same time (either in point-to-multipoint or mulitpoint-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.

When a session is established on peer-to-peer type networks, the Level I module that is responsible for initiating the session sends a **POLL** message to the local Level II module. When the Level II module receives the **POLL** message it scans the table of outgoing objects and finds the object that should be delivered to the peer device. For small mobile devices, the table of outgoing objects contains only one entry, so the selection of the object is trivial. For servers, the table will have multiple entries. Therefore, linear search is used to select the outgoing object. When the object is selected, the Level II module sends an **OBJ** message that contains the class of the object, its size and a time stamp, to the other end. On master-slave type networks the transmission procedure is slightly different. When the Level I module establishes a session, it sends a **HELO** message instead of **POLL** to the local Level II module, which then sends the **POLL** message as a response to **HELO** message to the other device.

Upon reception of an **OBJ** message, the device examines the class and time stamp on the object and decides to either accept or reject it. Objects are rejected if either the receiver does not accept the type of object that sender is trying to send or if the receiver already has a newer copy of the object of this type. If the device decides to reject this object it sends back a **REJ** message. If not, it sends back a **PROCEED** message that contains the bitmap of the already received segments of the object. For the very first session, this bitmap has all its bits set to 0.

C.3 Message Transmission

When the **REJ** message is received, the device removes the object from the table of outgoing objects. If this device acts as a slave, it sends a **NOPE** message to the master. This will cause sender and receiver to interchange their roles and repeat the session.

When the device receives a **PROCEED** message, it updates its copy of the bitmap and starts sending packets that correspond to the bitmap entries marked 0. The process of sending packets is as follows. After determining which message segment needs to be sent, the device prepares the packet with its initial header containing the string **PK** followed by the slot number and then the contents of the message. This packet is copied to the common communication area and the *out data* flag is set. The Level II state machine then switches the state to 'wait for **SENT**'. The Level I code picks up this data packet and sends it to the appropriate communication channel. When the **PK** message is received at the other end, that device copies the contents to the appropriate slot in the object reception buffer and sets the bit corresponding to this slot to 1.

For peer-to-peer network media that provide buffering of the outgoing packets, the Level I module sends a **SENT** message to the local Level II module right after the packet is placed on the network stack to transmit. For master-slave type networks, the Level I module places the packet on the network stack and then returns to the event loop of the application. When the network stack completes transmission of the packet it sends a *packet handled* indication to the Level I module, which in turn, sends the **SENT** message to the local Level II module. Reception of the **SENT** message by the Level II module causes the bit corresponding to the last transmitted segment to be set to 1. It then checks the bitmap and the object size to determine if it needs to send any additional packets. The same process applies to all messages that are exchanged between two devices. However, the synchronous nature of session set up makes the handling of the **SENT** message optional for the rest of the exchanged messages. If a control message is lost because of the problems with local network stack or during transmission, the whole session will be terminated by the watchdog timer.

C.4 Session Completion

After the sender completes packet transmission, it sends a **DONE** message to the other side and goes to the 'wait for message' state. When the **DONE** message is received at the other end the Level II module checks its own message bitmap. If all the expected message segments are received, it responds with the **ACK** message and updates the time stamp of the last received object in its object acceptance table. 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. If the receiving side gets the **DONE** message and discovers 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.

IV. Performance Metrics and Experimental Setup

A. TCP and CentaurusComm 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 records the time just before transmitting the request on the channel and again immediately after receiving the reply.
This is the only metric used to evaluate the performance of CentaurusComm. In order to obtain accurate RTT measurements on the CentaurusComm server, we recorded and analyzed the output produced by the **tcpdump** program.

- **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: (i) loss of signal between wireless client and base station and (ii) 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.

B. 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. 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.
- **Cell Busy**: This is a measure of the "busyness" of the cell that supports the client. 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.

C. General Experimental Setup

We used the typical client server scenario to perform experiments. The client program was executed 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 program on a Linux-based (version 2.2.17) system using the Bluetooth stack developed by Axis Communications, Inc. We used Bluetooth hardware developed by Ericsson. A concurrent server also executed on a Linux-based (version 2.2.17) system. In all experiments, the wireless client initiates the connection to the server that is part of a LAN.

C.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. The good-bye message consisted of all recorded values, of various parameters described above, on 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 Web browsers and E-mail. 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.)

C.2 Palm Pilot/CDPD Specific Setup

The experiments consisted of 100 sessions per variable packet size, which ranged from 128 bytes to 8192 bytes. Two sets of experiments were performed. In one set, the request size was varied and in the other, the reply size was varied. Thus, starting with the shortest size - 128 bytes - the request (reply) size was increased every 100 sessions in the first (second) experiment.

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. 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 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.

Another observation we made during testing is that TCP packets of size 8192 bytes cannot be sent from Palm Pilots successfully. The main problem is the small Maximum Segment Size (MSS) of 536 bytes, leading to fragmentation of large packets into 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 TCP on 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, invariably, that 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 the CentaurusComm protocol 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.

C.3 Bluetooth Specific Setup

As mentioned above, both the client and server programs were executed on Linux-based systems, to test TCP 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 ranging from 64 bytes to 8192 bytes. The experiment sets were generated in the same manner as for CDPD. However, in order to exchange TCP 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.

C.4 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/reply), connection setup mode (on/off), time-of-day, location** and **client mobility**. 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) possible 5-tuples. We allow each variable to take only a subset of the possible values and thus reduce the size of the space considerably.

V. Experimental Results and Discussion

A. Performance of TCP and CentaurusComm over CDPD

A.1 TCP over CDPD

Different locations were chosen to measure performance of TCP over CDPD. Each location experienced different signal strengths, peak hours and channel busyness. However, our analysis indicates that the overall performance of successful transmissions is quite similar. The graph in figure 1 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 bytes 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. In some cases they cause the interruption of the whole transmission process. This is a well-known problem discussed in detail in [17].

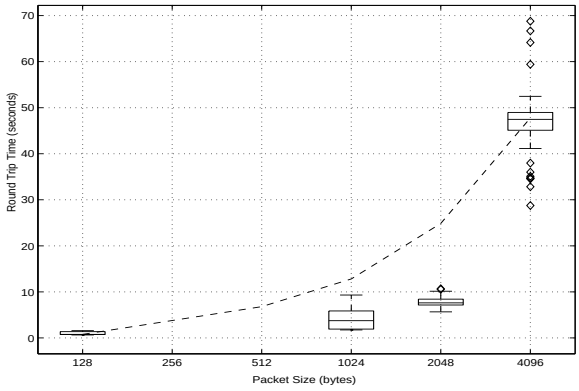

Fig. 1. Performance of TCP over CDPD

The graph in figure 2 shows the influence of the measured environmental factors on the performance of TCP for 1024-byte packets. The plots of the 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. "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 the figure is the difference between maximum and minimum RTT values for the chosen number of data points.

The graph suggests 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 figure 2, we see that a combination of moderate BLER and Cell Congestion lead to higher RTT and lower efficiency values.

A.2 CentaurusComm over CDPD

Experimental results indicate that the CentaurusComm protocol is not as prone to performance degradation as TCP over

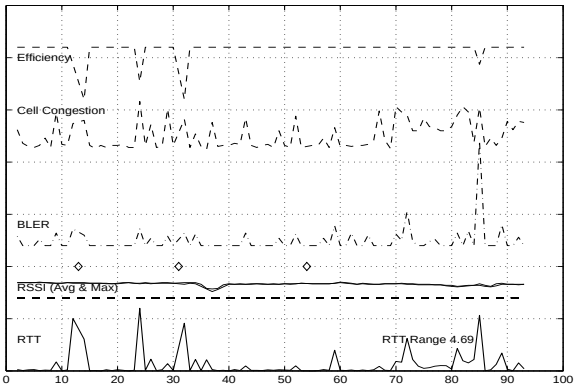

Fig. 2. Influence of environmental factors on TCP performance (downtown)

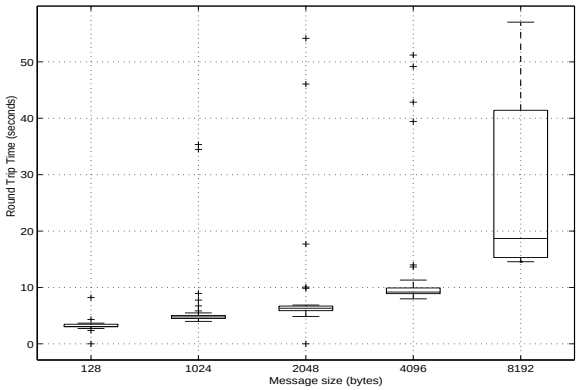

Fig. 3. Performance of CentaurusComm over CDPD

CDPD. The graph in figure 3 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 seconds. Therefore, 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 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. The graph in figure 4 shows the relationship between environmental conditions and RTT. The packet size in figure 4 is 8192 bytes. 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 seconds because of the initial and final synchronization of the sender and receiver. However, this could be improved by re-

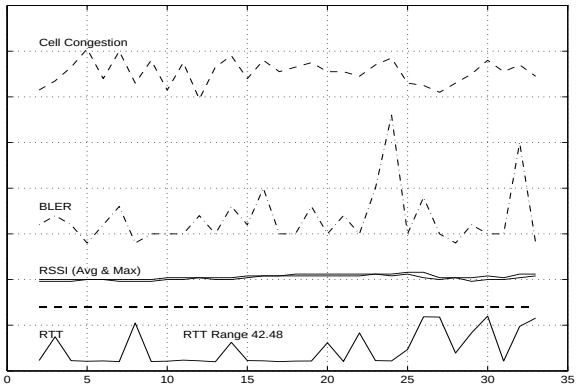

Fig. 4. Influence of environmental factors on performance of CentaurusComm in CDPD for large transmissions

designing the protocol so that it starts sending the beginning of the message before it gets the **PROCEED** message from the receiver. This would greatly improve the performance in cases when the environmental conditions are favorable, but would not significantly reduce the performance under unfavorable conditions.

B. Performance of TCP and CentaurusComm over Bluetooth

B.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 bytes 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 figure 5 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 [17].

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. We performed Bluetooth testing on Linux-based 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. Evaluating TCP over Bluetooth using smaller devices is part of our future work.

B.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

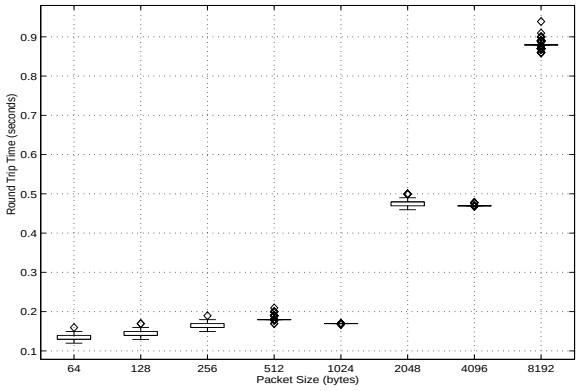

Fig. 5. Performance of TCP over Bluetooth

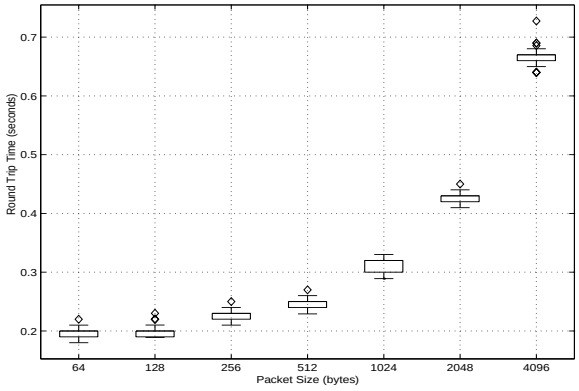

Fig. 6. Performance of CentaurusComm over Bluetooth

type of networks was used in this case as well. Analysis of CentaurusComm performance over Bluetooth shows that the protocol scales 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; the difference in RTT is around 50ms up to 512-byte packets. For packet sizes 1024 bytes and above, the difference in RTT ranges between 100ms and 150ms. We attribute this to the slower execution of segmentation and reassembly in CentaurusComm, in user space. 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 3.5s from an average of around 0.6s for 4096 byte packets. We are trying to determine the cause(s) of this behavior. The graph in figure 6 shows the performance of CentaurusComm over Bluetooth for different message sizes up to 4096 bytes.

VI. Conclusions and Future work

In this work we have conducted an empirical study of the performance of two different transport protocols over wireless networks. Simulation studies have shown that TCP does not perform well in wireless networks. However, we believe that our work is one of the few that consists of experimental evaluation and analysis of TCP in wireless networks. We have also described CentaurusComm, a protocol optimized for wireless networks, evaluated its performance *empirically* and compared it with TCP. We also conclude that results obtained empirically are very important in evaluating performance of protocols in wireless networks. In the future we are planning to conduct more rigorous testing of these protocols on Bluetooth and other types of wireless networks. Future work also includes clean integration of CentaurusComm over the Bluetooth Stack. The current method of sending CentaurusComm messages over the PPP/RFCOMM/L2CAP part of the stack is quite expensive in terms of time. We are now working on integrating CentaurusComm directly with the HCI and the L2CAP layers.

References

[1] "The official Bluetooth SIG website," `http://www.bluetooth.com`.

[2] "Wireless Ethernet," `http://www.wirelessethernet.com`.

[3] M. Taylor, W. Waung, and M. Banan, *Internetwork Mobility: The CDPD Approach*, Prentice-Hall Inc., 1996.

[4] "GSM World," `http://www.gsmworld.com`.

[5] J. Ahn, P. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: emulation and experiment," in *Proc. ACM SIGCOMM*, 1995.

[6] K Fall and S Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[7] C. Fang, H. Chen, and J. Hutchins, "A simulated study of TCP performance in ATM networks," *ATM Forum Contribution 94-0119*, 1994.

[8] J. Mo, R. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in *Proc. IEEE INFOCOM*, 1999.

[9] J. Postel, "Transmission Control Protocol," *Network Information Center RFC 793*, pp. 1–85, 1981.

[10] "The Network Simulator - ns-2," `http://www.isi.edu/nsnam/ns/`.

[11] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. ICDCS*, 1995.

[12] K. Brown and S. Singh, "A network architecture for mobile computing," in *Proc. IEEE INFOCOM*, 1996.

[13] B. Bakshi, P. Krishna, N. Vaidya, and D. Pradhan, "Improving performance of TCP over wireless networks," in *Proc. ICDCS*, 1997.

[14] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE Journal on Selected Areas in Communications*, 1995.

[15] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, 1997.

[16] K. Ratnam and I. Matta, "WTCP: An efficient mechanism for improving TCP performance over wireless links," in *Proc. ISCC*, 1998.

[17] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "WTCP: A reliable transport protocol for wireless wide-area networks," in *Proc. ACM MOBICOM*, 1999.

[18] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, and A. Joseph, "Multi-layer tracing of TCP over a reliable wireless link," in *Proc. ACM SIGMETRICS*, 1999.

[19] B. Rathke, M. Schlager, and A. Wolisz, "Systematic measurement of TCP performance over wireless LANs," Tech. Rep., Technical University of Berlin, Germany, 1998.

[20] Aether Systems Inc., "Aether Intelligent Messaging," .