Wireless Networks 8, 301–316, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks

PRASUN SINHA
Bell Laboratories, Lucent Technologies, New Jersey, USA

THYAGARAJAN NANDAGOPAL
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, USA

NARAYANAN VENKITARAMAN
Motorola Labs, Schaumburg, IL, USA

RAGHUPATHY SIVAKUMAR
Electrical and Computer Engineering, Georgia Institute of Technology, USA

VADUVUR BHARGHAVAN
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, USA

Abstract. Wireless wide-area networks (WWANs) are characterized by very low and variable bandwidths, very high and variable delays, significant non-congestion related losses, asymmetric uplink and downlink channels, and occasional blackouts. Additionally, the majority of the latency in a WWAN connection is incurred over the wireless link. Under such operating conditions, most contemporary wireless TCP algorithms do not perform very well. In this paper, we present WTCP, a reliable transport protocol that addresses rate control and reliability over commercial WWAN networks such as CDPD. WTCP is rate-based, uses only end-to-end mechanisms, performs rate control at the receiver, and uses inter-packet delays as the primary metric for rate control. We have implemented and evaluated WTCP over the CDPD network, and also simulated it in the *ns-2* simulator. Our results indicate that WTCP can improve on the performance of comparable algorithms such as TCP-NewReno, TCP-Vegas, and Snoop-TCP by between 20% to 200% for typical operating conditions.

Keywords: Wireless transport, WTCP, reliable wireless transmission

1. Introduction

Recent years have witnessed an explosive growth in the use of wireless wide-area networks (WWANs) such as CDPD [23], RAM [23], Ardis [23] and HDR [22], with industry projections topping \$2.5 billion by the year 2002. Many large corporations are equipping their mobile work-force with laptops that have WWAN connectivity, thus enabling users to be connected anytime, anywhere. In the typical WWAN deployment scenario, a mobile user connects over the WWAN network to a dedicated proxy in the corporate backbone, which then acts as the service point for all the requests from the mobile user[1]. Providing efficient and reliable connectivity between the proxy and the mobile host over commercial wide-area wireless networks is, thus, becoming a critical issue.

[1] Most of the current deployment uses the proxy model for three reasons: (a) the connection quality of the WWAN network is too poor to sustain typical client-server applications, (b) the amount of data transferred to the mobile host must be filtered because of the orders of magnitude difference in bandwidth between the wired and WWAN connections, and (c) many portable computing devices have display and processing limitations that must be addressed by the proxy before sending the processed/filtered response to the mobile user.

Despite the enormous commercial interest, a typical WWAN network exhibits some or all of the following characteristics: very low and highly variable throughput (between O(100 bps)–O(10 Kbps)), very high and highly variable latency (between O(400 ms)–O(4 s)), bursty and random packet losses unrelated to congestion (1–10%), and occasional blackouts, some of which can exceed 10 s. Under such operating conditions, applications require a generic transport protocol that is amenable to different types of wireless networks which exhibit similar characteristics. However, standard transport protocols such as TCP perform very poorly in such conditions because of two main reasons: first, TCP assigns all packet losses to congestion and throttles down the transmission rate upon detecting a packet loss; and second, TCP sets its retransmission timeout (RTO) based on observed round trip times ($\mathrm{RTO} = (\overline{rtt}) + 4 \times \Delta(rtt)$). For the WWAN networks under consideration, neither mechanism is appropriate. Packet losses may happen either due to adverse channel conditions or congestion, and throttling down the transmission rate in the former case is the wrong thing to do. Furthermore, due to the very low bandwidth, transmission delays contribute significantly to the observed round trip time. Thus, round trip time computations are highly susceptible to fluctuation depending on the size of the TCP congestion window. As a consequence, RTO values computed by the sender tend to be very high – as large as 32 s for CDPD. A combination of large RTO for loss recovery and wrongly throttling the congestion window upon random packet loss can result in extremely poor TCP performance under typical WWAN operating conditions, as we have observed in our experiments with the CDPD network.

In recent years, there have been several proposals to optimize TCP for wireless networks. These proposals typically have had three flavors: (a) improving reliability at the link layer [10], (b) providing TCP-aware smarts in the base station [4], and (c) splitting the TCP connection into two parts [2,24] – one in the backbone network and one over the wireless link running a specialized protocol [24]. Most of these solutions focus almost exclusively on making the loss characteristics of the wireless link transparent to the sender, while preserving the congestion control mechanisms of TCP. As we will discuss in section 2, contemporary solutions address only a part of the problem of WWAN environments and leave a lot of scope for improvement.

In this paper, our goal is to identify the fundamental causes of performance degradation of TCP in commercial WWAN networks, and design a transport protocol that effectively addresses these causes. We seek to design a deployable, robust, fair, efficient, and reliable transport protocol for commercial WWAN networks. To this end, we present the *Wireless Transmission Control Protocol* (WTCP), which enunciates the following three key principles:

1. WTCP uses purely *end-to-end mechanisms*, thereby eliminating the need for transport-level support from the network infrastructure, in contrast to related work [2,4,10,24]. We choose end-to-end mechanisms for three reasons. First, an end-to-end solution generically aimed for all wireless networks would eliminate the need for network specific transport layers at the end hosts. For the end host, especially the oft resource-scarce mobile host, this results in a single transport layer solution for all wireless networks and hence, allows the end hosts to move seamlessly across different wireless networks. Second, the connection end-points in most of the current WWAN deployment scenarios are a mobile host and a dedicated proxy – both of which are already wireless/mobility aware. By deploying WTCP at the mobile host and the proxy, it is thus feasible to propose end-to-end transport layer solutions for WWAN environments without having to change the TCP implementations on all stationary hosts. Third, WWAN network providers are typically unwilling to introduce TCP-aware smarts in the base station or mobile switching station for reasons discussed in section 2.
2. WTCP uses *rate-based* rather than window-based transmission control. As a result, WTCP shapes its data traffic, never allows a burst of packet transmissions, and is also less prone to the unfairness observed in TCP when competing connections have different round-trip times [16]. Furthermore, it is the receiver that adaptively computes the desired transmission rate based only on the characteristics of the data path. Consequently, WTCP is relatively insensitive to problems in the ACK path and can handle asymmetric channels well.
3. WTCP uses the ratio of the *inter-packet separation* at the receiver and the inter-packet separation at the sender (recall that the sender shapes data traffic) as the primary metric for rate control rather than using packet loss and retransmit timeouts. It also tries to remain in congestion avoidance phase at all times by detecting and reacting to incipient congestion. As a result, WTCP reduces the effect of non-congestion related packet losses on the computation of transmission rate. Additionally, WTCP is less prone to large round-trip-time variations as the congestion window size increases, which is inevitable in standard TCP-variants due to the low-bandwidth nature of the channel.

We do not claim that WTCP is the manna for wireless transport protocols; as with any other transport protocol, it has its own disadvantages. In this paper, we also present a detailed discussion on the issues regarding the design and implementation of WTCP.

The rest of the paper is organized as follows. Section 2 identifies the typical characteristics of the WWAN environment, and how they adversely affect the performance of TCP. Section 3 describes the key design principles and mechanisms of WTCP. Section 4 presents the algorithms for congestion control and reliability for WTCP. Section 5 evaluates the performance of the WTCP through a user-level implementation tested over CDPD networks and through simulations in *ns-2*. Section 6 compares WTCP to related work. Section 7 presents some design issues and future work in WTCP. Section 8 concludes this paper.

2. Characteristics of WWAN environments

At a high level, most WWAN environments exhibit one or more of the following characteristics: the bandwidth is very low and varying, the latency is very high and varying, blackouts exceeding 10 s occur occasionally, and for unreliable WWAN networks, non-congestion related packet loss may be significant when the user is traveling at moderate to high speeds.

We have used CDPD as the evaluation platform for WTCP though our design is not specific to any particular data network. CDPD is a packet data network that overlays the AMPS cellular telephone infrastructure. A CDPD "full duplex channel" is a pair of unidirectional channels, each with a raw capacity of 19.2 Kbps. Up to maximum of 30 users can share a pair of uplink/downlink channels. A set of channels may be dedicated for CDPD transmission, or CDPD users may be dynamically assigned to channels that are preferentially shared by cellular phone calls – in the latter case, users are more likely to see short-term blackouts. CDPD transmits compressed and encrypted data, and adds 48.2% Reed–Solomon coding overhead for forward error correction. Two characteristics of CDPD are germane to our discussion: the effective throughput of a CDPD channel typically does not exceed 12 Kbps, and the majority ($\geqslant$75%) of the end-to-end latency is incurred in the CDPD part of the network between the mobile switching station and the mobile host. We will revisit the impact of the latter point when we discuss wireless transport protocols that rely on TCP-aware smarts at the base station [4].

WWAN wireless networks in general, and CDPD networks in particular, typically exhibit the following six characteristics.

1. *Non-congestion related packet loss.* Even though CDPD adds 134 bits of Reed–Solomon error-correcting code to every 278 bit block of data, we have measured non-congestion related error ranging from 0 to 10% depending on the speed of mobility (measured over a range of 0–55 mph), location of the user vis-à-vis the base station, and co-channel interference.
   TCP assumes that all packet losses result from congestion. A 5% non-congestion related packet loss can, thus, significantly degrade the performance of TCP.

2. *Very low bandwidth.* As we mentioned above, between 1 to 30 users may share a single CDPD channel of raw capacity 19.2 Kbps. For RAM, the channel is 8 Kbps, while for Ardis, the channel may be 4.8 Kbps or 19.2 Kbps.
   Due to the extremely low bandwidth, the delay-bandwidth product of a connection is small (typically 2 or 3 packets). This can affect the congestion control and fast retransmit mechanisms of TCP adversely. TCP sometimes observes artificially larger congestion windows as a result of deep buffering in the CDPD network. While this allows a connection to pump in more packets into the network, it artificially increases the round trip time and significantly affects TCP performance in case of a timeout.

3. *Large round trip time and variance in round trip time.* In CDPD, we have observed typical round trip times between 800 ms to 4 s. A large fraction of this time is due to transmission on the wireless link (e.g., transmitting a 512 byte packet at 12K bps takes 300 ms), and over 75% of the latency is typically incurred in the segments of the connection lying in between the mobile switching station and the mobile host. In figure 3(a) we observe the variation in round trip time for UDP handshakes that progress in bursts of 8 from 1.8 s to 6 s for successful handshakes. This plot shows the impact of large transmission delay on *rtt* when the sender bursts packets.
   TCP sets the retransmission timeout to be the sum of the average round trip time and four times the mean deviation of the round trip time (RTO $\leftarrow \overline{rtt} + 4 \times \Delta(rtt)$). This may result in very large RTOs (e.g., 32 s) in CDPD because of two reasons: (a) *rtt* and $\Delta(rtt)$ are inherently large, and (b) ACKs from the receiver get bunched (see below), and since ACKs clock data packets in TCP, data packets are sent out in bursts, which further increases the mean and deviation of *rtt*. Thus, timeouts affect TCP performance very severely on CDPD.

4. *Asymmetric channel – bunching of ACKs.* CDPD uses DSMA/CD [1] for contention resolution in the channel. Contentions among mobile users for the uplink channel are resolved by binary exponential backoff. Consequently, CDPD suffers from the well known “capture syndrome” of binary exponential backoff [21], in which a highly loaded shared medium ends up bursting the queued packet transmissions of each contending host in turn. RAM also suffers from the same problem.
   For the common case of downlink data transmission, ACKs from the mobile to the backbone host get bunched. This further skews the round trip time computation, and also causes the sender to burst out packets as mentioned above.

5. *Occasional blackouts.* Prolonged fades, sudden degradation in signal quality such as traveling through a tunnel or between overlapping base stations, and temporary lack of available channels (when cellular phone calls are occupying the channels) can cause blackouts lasting 10 s or more, and results in the back-to-back loss of a sequence of packets. Traveling at 55 mph, we observed several blackouts ranging from 10 s to 10 min during the course of a day.

6. *Inter-packet delays as a congestion metric.* We have observed that sharp increases in the short-term average interpacket delay observed by the receiver almost always precede congestion-related packet loss in the CDPD network. *Specifically, an increase in the average interpacket delay perceived by the receiver is an indication of increased contention for the wireless link and is a precursor to loss unless connections throttle back their sending rate.* In figure 3(b) we observe the interpacket delay sample points, the computed long-term and short-term interpacket delay averages, and the computed mean deviation as a function of time at the receiver. The sender sends packets with a constant interpacket separation of 1 s. Clearly, interpacket separation is a useful metric to pace the progress of the connection and can be used to perform rate control.
   In wireline networks, the use of both delays and interpacket separation as a metric for predicting congestion has not been well accepted because of the large variation in delays experienced by packets over the Internet. However, this approach works well in our target environment because of the extremely low bandwidth of WWANs, wherein the transmission time over the wireless link predominates. TCP-Vegas uses a variant of this approach by monitoring round trip times at the sender instead of interpacket delays at the receiver [6]. Unfortunately, using TCP-Vegas as-is will not work as well in WWAN environments because of asymmetric channels and the effect

of bursting packets on the computed *rtt* (points 2 and 4 above).

2.1. High level architectural tradeoffs in WWAN environments

Before going into the details of the WTCP design, we need to step back and discuss the tradeoffs between using end-to-end mechanisms for wireless TCP versus using smart mechanisms in the network in order to assist wireless-unaware TCP end-points. Related work on wireless optimizations for TCP has typically argued against end-to-end mechanisms on the grounds that it is impractical to change the protocol stacks of all stationary hosts merely to accommodate mobile hosts. Thus, most of the previous work has focused on making the lossy nature of the wireless link transparent to the stationary end host by introducing smarts at the base station via one of three mechanisms: reliable link layers [10], TCP-aware "snoop" mechanisms [4], or splitting the connection into two (wireline and wireless) distinct components [2,24].

Link layer retransmission [10] works well when the latency over the wireless link is small compared to the coarse-grained TCP timer. In the ideal case, the link layer retransmissions are not expected to significantly interfere with the end-to-end *rtt* computations or congestion control mechanisms – except to eliminate random channel loss. In WWAN networks, it is the transmission time over the wireless network that constitutes the bulk of the observed end-to-end latency. Consequently, providing only a reliable link layer abstraction at a packet-level time scale and keeping TCP unchanged at the end hosts simply will not work because they will interfere with the reliability and congestion control mechanisms of TCP. If the wireless data network provides fine grained link-level retransmissions, e.g., RLP used in HDR [22], then link layer mechanisms can effectively mask channel error from higher layers.

The Snoop protocol [4] instantiates TCP-aware smarts at the base station (or mobile switching station) in order to eliminate the problem of false fast retransmits or slow starts due to random packet loss over the wireless channel. This approach assumes that the transmission time over the wireless link is significantly smaller than the coarse-grained TCP timer and round trip time. Moreover, Snoop works well only when the bandwidth-delay product of the wireless link is at least 3 packets long. However in WWAN environments, Snoop does not work well because of two reasons: (a) it exacerbates the problem of large and varying round trip times by suppressing duplicate ACKs, and (b) duplicate retransmissions may be initiated by both the Snoop agent and the end host (which may observe a timeout) because of comparable timeout values at the two entities. In fact, we have observed that snooping may possibly degrade the performance of TCP when the latency over the wireless link dominates the round trip time. In summary, we believe that Snoop works well in the environment for which it was designed, but it does not work well in the WWAN environment.

Indirect TCP protocols [2,24] break the TCP connection at the base station, and maintain two separate connections – one over the wireline network and one over the wireless network. I-TCP violates the fundamental end-to-end guarantees of TCP by splitting the connection. Note that the connection split must happen at the base station (or mobile switching station) serving the mobile host, and the connection state must be moved across base stations upon handoff. The wireless component of I-TCP is quite simplistic and does not address issues such as (a) non-conformance with end-to-end semantics, (b) overhead of moving state between base stations, (c) deployability constraints due to mandatory changes in base stations, and (d) design of the transport protocol for the wireless link. As a practical matter, the I-TCP architecture may not be feasible for WWANs because it requires significant infrastructure support and maintenance of connection state from the WWAN network, which is autonomously managed and may not even understand TCP/IP internally (e.g., RAM). It is important to note that I-TCP calls for splitting each transport connection transparently to the TCP end-points. It is, thus, quite distinct from the WWAN deployment scenario of a mobile host connecting to a dedicated stationary proxy in the backbone.

To summarize, we believe that previous approaches that seek to hide the problems of WWAN networks from TCP at the end host by adding TCP-aware smarts in the mobile switching station are not applicable to WWAN networks for two reasons: (a) such approaches require the base station to maintain significant state, understand TCP/IP, and are often tuned to specific flavors of TCP, and (b) the fact that the latency between the mobile switching station and the mobile host is the dominant component of a large and varying round trip time makes such approaches less effective. The bottom-line is that for WWAN environments, both endpoints must cooperatively address the issues unique to the environment. Moreover, it is desirable to eliminate network-level smarts because the base stations are owned by an autonomous entity that may not even be running IP internally. We believe that the key issues that need to be addressed – the non-congestion related packet loss, large and highly varying latency, asymmetry in data/ACK channel behavior – can be effectively solved with the end-to-end mechanisms proposed in this paper. Of course, the penalty for using the end-to-end mechanism is that the remote end-point in the backbone must also change. Luckily, the nature of the WWAN environment and the current deployment pattern already supports the common case of WWAN users typically connecting through a dedicated proxy server on the backbone.

3. The WTCP approach

Any reliable transport protocol must provide the following functions: (a) connection management, (b) congestion control, (c) flow control, and (d) reliability. WTCP reuses the standard TCP mechanisms for flow control and connection management. We now focus on the key design choices in WTCP for congestion control and reliability.

3.1. Congestion control

The key aspects of congestion control in WTCP are that it is rate based, uses interpacket delay as the primary mechanism to determine rate adaptation, performs the rate adaptation computations at the receiver, predicts the cause of packet loss and reacts accordingly, and varies the granularity of rate increase/decrease depending on the type of congestion observed. Additionally, WTCP also tailors its startup behavior to work well for short-lived flows. We describe the design aspects of WTCP congestion control in more detail below.

1. *Rate-based transmission control.* As we mentioned in section 2, for the common case of bulk data transfer from the backbone host to the mobile host, ACKs are often bunched together on the return path to the sender because of the nature of channel arbitration, e.g., DSMA/CA used in CDPD. With the window-based self-clocking mechanism of TCP, this results in the sender bursting back-to-back data packets, which skews round trip time computations, causes more bursty queuing at the base station, and consequently more packet drops. WTCP alleviates these problems by using a rate based scheme that does not use ACKs for self-clocking. Adopting a rate-based approach does involve explicit clocking and shaping the traffic according to the current transmission rate of the connection; however, the rates are typically small enough that coarse grain timers (O(100 ms)) are sufficient to perform the clocking effectively.

2. *Inter-packet delay as the main mechanism for transmission control.* We have observed that monitoring the average interpacket delay at the receiver provides a fairly accurate measure of the available channel rate for low bandwidth channels. Specifically, the ratio of the average inter-packet separation at the receiver and the average inter-packet separation at the sender provides a responsive metric to determine if the transmission rate needs to be increased, or decreased. Thus, when the network is uncongested or has incipient congestion, reacting to changes in the interpacket delay ratio serves to keep the network uncongested, and significantly reduces the number of congestion-related packet losses. WTCP, thus, uses this mechanism as the primary transmission rate control mechanism, and essentially uses incipient congestion detection without waiting to lose packets before throttling down the sending rate.

3. *Distinguishing the cause of packet loss and adjusting transmission rate accordingly.* While interpacket delays are the main mechanism for dealing with incipient congestion, if the network suddenly moves from uncongested to congested state (e.g., due to a sudden influx of new connections or sudden decrease in available resources), then packets are dropped due to congestion. Our algorithm must detect such losses and throttle the sending rate aggressively. In other words, when the receiver observes packet losses, it must predict the cause of the losses and react appropriately. If the loss is predicted to be due to congestion, then the sending rate is throttled down.
   We use a heuristic based on the average per-packet separation to distinguish congestion losses from random losses. In this heuristic, the receiver initially predicts that all losses are non-congestion losses. Consider that packets i and j were received ($j > i$), but packets $i+1, \ldots, j-1$ were all lost. In this case the receiver computes the average inter-packet separation for each of the lost packets equal to

   $$per\_pktsep \leftarrow \frac{recv\_time_j - recv\_time_i}{j - i},$$

   where $recv\_time_i$ is the time at which the last bit of packet i arrives. If the value of *per_pktsep* is close to the measured inter-packet separation at the receiver (i.e. within the band [average $-$ K $\cdot$ mean deviation, average $+$ K $\cdot$ mean deviation], where K is a constant), then the receiver predicts that the losses were all random losses. Otherwise, the receiver predicts that there was at least one congestion loss, and the sending rate is reduced by half.

4. *Performing transmission control computations at the receiver.* In WTCP, the receiver performs the rate control mechanisms described above, and computes the new transmission rate of the sender. With each data packet, the sender transmits its current interpacket separation. Based on local state and the state in the packet, the receiver has all the information it needs to update the transmission rate. This is done at regular intervals, which we refer to as *update epochs*. An update epoch begins when the sender starts using a new rate and lasts for a fixed time period. At the end of the epoch, the receiver performs the rate control computations and sends the rate update back to the sender in its acknowledgment. Having the receiver perform the rate computations eliminates the effect of delay variations and losses in the ACK path. Even if ACKs get bunched, delayed, or lost, the transmission rate is not altered. WTCP can thus deal with asymmetric channels effectively.

5. *Variable granularity rate adjustment.* TCP uses the well known linear-increase–multiplicative-decrease policy for adjusting its congestion window. While LIMD is stable and asymptotically converges to fair channel allocation, the efficiency of the LIMD algorithm is a function of how severely the decrease is performed. TCP reduces its congestion window by half upon observing a packet loss. In WTCP, we seek to detect incipient congestion and react to it early on in the common case. The goal of WTCP is to decrease the transmission rate multiplicatively in order to ensure fairness, less aggressively when reacting to incipient congestion in order to improve efficiency, and more aggressively when reacting to real congestion in order to reduce packet loss and alleviate congestion quickly. In order to achieve these goals, WTCP maintains a history of transmission increase/decrease in the recent past. If

the receiver is required to perform transmission decrease multiple times in quick succession, it starts to decrease its transmission rate more aggressively. If the receiver observes a congestion based packet loss, it halves its rate. As a result of this approach, incipient congestion is handled by a gentle decrease of the transmission rate, but severe congestion is handled by an aggressive decrease in transmission rate.

6. *Startup behavior.* Since round trips are large in WWAN environments, and since some data transmissions may be short-lived, WTCP attempts to compute the appropriate transmission rate for a connection immediately upon startup rather than going through slow start. WTCP uses the "packet-pair" approach [13], wherein it sends two back-to-back packets of maximum segment size (MSS) and computes their interpacket delay during connection establishment. This serves as an approximate estimate for the sending rate. Though the packet-pair approach is known not to work too well in wireline environments [20], we have used it as a first-cut approach. We will investigate this approach in the near future.
7. *Blackout detection.* Blackouts occur when the connection experiences back-to-back losses for extended periods of time due to poor channel conditions or lack of available channels. As described in section 3.2, the reliability mechanism of WTCP elicits an acknowledgment (positive or negative) from the receiver for every packet that has been sent, before the sender decides to resend it. Thus, if a packet has not been acknowledged (positively or negatively) for a threshold period of time, the sender enters the blackout phase and starts probing the receiver in order to elicit an acknowledgment. Upon a successful packet handshake after entering the blackout phase, the sender reverts to the old transmission rate that it was using before the onset of the blackout phase. This ensures that the packet losses during the blackout phase do not affect the transmission rate.

3.2. Reliability

The key aspects of reliability in WTCP are that it uses selective acknowledgments, it does not use retransmit timeouts, and that it tunes the frequency of sending acknowledgments to the dynamic network conditions. We describe these aspects below:

1. *Selective acknowledgments.* As noted in related work, selective acknowledgments are very useful in TCP [17]. WTCP uses selective acknowledgments for ensuring reliability. The receiver periodically sends ACKs at a frequency tuned by the sender (see below), containing the cumulative and selective ACK. By inspecting the ACK, the sender can detect a hole in the receiver's sequence of received packets. By comparing the state contained in the ACK with local state stored with the last (re)transmission for each unacknowledged packet, the sender can determine if this last (re)transmission was lost, or could still be in transit. Thus, selective acknowledgment allows the sender to retransmit only lost packets.
2. *No retransmit timeouts.* As we have observed in section 2, it is exceedingly difficult to maintain a reliable estimate of the retransmit timeout. In fact, many of the performance problems observed in various TCP flavors are caused by erroneous RTO estimation. WTCP does not use RTOs. Instead, it modifies the SACK algorithm in order to achieve reliable transmission without RTOs. This mechanism is described in section 4, and is a very important aspect of WTCP.
3. *Controlling ACK frequency.* ACKs carry both reliability and transmission control information, and the sender must receive ACKs periodically in order to react to the new transmission rate, and perform flow control. The sender tunes the desired ACK frequency (and notifies the receiver in the data packet) such that it expects to receive at least one ACK in a threshold period of time (e.g., 5 s). If the sender has one or more packets pending acknowledgment for more than a threshold period of time, it goes into the blackout mode. The tuning of the ACK frequency is governed by several factors: (a) observed ACK loss at the sender, (b) half-duplex or full-duplex nature of the WWAN channel, and (c) average and deviation in the inter-ACK separation observed at the sender. Note that a receiver may also voluntarily generate a SACK immediately upon observing a hole in the packet sequence. The effects of these factors on controlling the acknowledgment frequency is part of ongoing work.

4. The WTCP algorithm

In WTCP, *rate control and reliability are decoupled*. The receiver computes the desired sending rate via its rate control mechanisms, and notifies this rate to the sender in the ACK packets. ACKs, thus, carry both reliability information (SACK) and rate control information. The sender monitors the reception of ACKs, and adjusts its rate accordingly. It also monitors the ACKs to tune the ACKing frequency, which it then notifies to the receiver in future data packets. If the sender has one or more packets pending acknowledgments for more than a threshold period of time, it goes into blackout mode and periodically sends probe packets to elicit ACKs from the receiver and recover from the blackout. The probe packet mechanism is also used for loss recovery, eliminating the need for timeout-based retransmission in WTCP.

We now describe the parameters contained in the packets of WTCP, and then describe the key functions of WTCP. There are three types of packets: Data, ACK, and Probe. The contents of these packets are as follows:

Data: ⟨*rel_seq_num, cctrl_seq_num, hack_seq_num, current rate, ack frequency, packet_size, data*⟩.

ACK: ⟨ *ack_seq_num, hcctrl_seq_num, updated rate, CACK, SACK* ⟩.

Probe: ⟨*cctrl_seq_num, optional_data*⟩.

The reliability sequence number (*rel_seq_num*) in the data packet is the byte sequence number as in TCP, and is used for sequencing and reliable delivery of packets. The congestion control sequence number (*cctrl_seq_num*) is a monotonically increasing packet sequence number that distinguishes multiple transmissions of the same packet. The ACK packet contains the CACK and SACK that acknowledge maximal sequences of contiguous data packets that have been received at the receiver, and also the highest congestion control sequence number (*hcctrl_seq_num*) seen thus far by the receiver. This is used by the sender in a variant of the SACK algorithm to eliminate retransmit timeouts.

ACKs are sequenced by the ACK sequence number (*ack_seq_num*), which is monotonically increasing for every ack sent from the receiver. Each data packet from the sender contains the highest ACK sequence number (*hack_seq_num*) that it has seen so far, and this information is used at the receiver in rate control to determine if a sender has already reacted to its previous rate adjustment.

Other parameters in the data packet include the current sending packet separation, the desired ACK frequency, packet size, and data. Besides other parameters, the ACK packet always contains the desired sending packet separation.

The probe packet contains a congestion control sequence number (*cctrl_seq_num*) but no data. As in the case of data packets, sending a probe packet also increments the *cctrl_seq_num* by 1.

We now describe the rate control and reliability mechanisms of WTCP.

4.1. Rate control

The choice of the threshold used for deciding whether to increase or decrease the transmission rate is crucial to the design of WTCP, and so is the use of history for computing the amount of decrease. These two key aspects of the rate control mechanism are described below:

- *Increase/decrease phase.* Recall that rate updates are epoch-based. The increase phase is used to probe the connection for available capacity. In our algorithm, we maintain a running average of the ratio of receiver to sender inter-packet separation, and update this average at the end of each epoch. A natural question arises at this point: given the running average of the ratio, what is the heuristic to determine whether the rate must be increased or decreased for the next epoch? In order to motivate this heuristic, let us now consider a single flow traversing a link of capacity, C. Let the rate of the flow be denoted by r. When $r < C$, the link can support the rate r, and hence, the ratio of the sending-to-receiving rate is 1. Thus, the receiver can increase the sending rate to probe the network for additional bandwidth. Note that the maximum value of r that the link can support is C. If the rate is increased beyond this value such that $r = C + \alpha$ (where α is the constant of increase), the maximum rate at which the receiver will receive is still C. The ratio of the sending rate to the receiving rate is, therefore, $(C + \alpha)/C$, which translates to $1 + \alpha \times$ *receive_separation*, where *receive_separation* is the time interval between arrivals of last bits of consecutive packets. The flow has to decrease its rate since the link is not able to support the rate of the flow. Thus, $(1 + \alpha \times$ *receive_separation*$)$ defines the threshold for increase or decrease for a single flow. The constant α should be chosen to be a small percentage of the capacity C. For our experiments, we chose α to be the number of bits in one maximum sized packet per RTT, thus making WTCP equally aggressive as TCP, to obtain a fair comparison with TCP.
- *Using history for graded multiplicative decrease.* Since we start to throttle the sending rate as soon as the ratio of the receiver to sender inter-packet separation exceeds the threshold, we react quickly to incipient congestion, and rarely get to the stage where packets must be dropped at routers due to congestion[2]. Since we expect to be in the congestion avoidance phase most of the time, we can afford to throttle more gently upon incipient congestion (rather than the traditional 50% in TCP). However, as congestion builds up, we must throttle rate more aggressively in order to quickly account for the built-up congestion and prevent packet losses from occurring. Essentially, this motivates having a simple notion of "history" of rate control and a simple gradation for rate throttling, where we progressively throttle rate more severely for back-to-back decrease phases. In our algorithm, we do this by maintaining a *decrease variable*, δ, which is reset to 0.1 whenever we encounter the increase phase, and is doubled in each decrease phase. During the decrease phase, the rate is throttled to a fraction $(1 - \delta)$. This provides a graded multiplicative decrease, as the granularity of decrease increases exponentially for successive decrease phases (i.e. if the previous decrease phase was not sufficient to throttle the rate adequately).

  As a result of this approach, incipient congestion is handled by a gentle decrease of the transmission rate, but severe congestion is handled by an aggressive decrease in transmission rate, which is not more than 50%. When a loss occurs, we reduce the sending rate by 50%, following popular TCP-friendly LIMD (Linear Increase Multiplicative Decrease) congestion control. In a related work, we have explored this algorithm in more detail [14].

Thus, the basic tenets of our approach are that it is rate-based with receiver-based rate computations, and that it uses linear increase and graded multiplicative decrease for congestion avoidance. With this framework in mind, we will now present the details of the rate control algorithm.

4.2. The rate control algorithm

The pseudo-code for the rate control algorithm is shown in figure 1. The rate control algorithm has two events associated

[2] This is true for wireless networks with a large delay component only if we assume deep buffering within the network. Our studies have shown that this is indeed true of typical wireless data networks.

```
    process_data_packet()
1   if (packet is out of sequence)
2       check_packet_loss()
3   else
        /* get current sample for recv_sep */
4       current_pktsep ← current_time - prev_pkt_time
5       if (sender has started using the latest computed rate)
6           ratio ← current_pktsep / send_packet_sep
7           aggregate_ratio ← aggregate_ratio + ratio
8           num_samples ← num_samples + 1
9   prev_pkt_time ← current_time

    update_timer_wakeup()
10  if (there were no samples in last epoch)
11      return
12  savg_ratio ← aggregate_ratio/num_samples
13  lavg_ratio ← exponential average of savg_ratio over last 6 epochs
                 with weight of 0.125 for new sample
14  num_samples ← 0
15  aggregate_ratio ← 0
16  if (lavg_ratio < 1 + $\alpha \times$ savg_ratio × send_packet_sep)
                 /* ← savg_ratio × send_packet_sep is the average receive_packet_sep */
17      rate ← rate + $\alpha$ /* Increase */
18      $\delta$ ← 0.1 /* reset decrease variable */
19  else
20      rate ← rate × (1 - $\delta$) /* Decrease rate by $\delta$% */
21      $\delta$ ← $min(2\delta, 0.5)$
```

Figure 1. Algorithm for rate control at receiver.

with it: the arrival of a packet at the receiver, and a periodic timer-based update.

When a packet arrives at the receiver, the receiver checks to see if the packet is in sequence. If the packet is out of sequence (line 1), the receiver assumes that all the intermediate packets have been lost. receiving N out of order packets). In the event of such losses, the receiver predicts if the losses are due to congestion and if so, it reduces the rate by half, or else, it treats the losses as random losses and does not react to them (line 2). This loss predictor uses the per-packet separation based heuristic described in the previous section. If the packet has been received in sequence, the receiver computes the separation between this packet and the previous packet (line 4).

If the packet has the updated rate, then the ratio of the receiving to sending separation is computed, and the aggregate of ratio over all samples is used to compute the short term and long term averages at the end of an update epoch (lines 6–8).

At the end of every update epoch, the short term average (*savg_ratio*) of the ratio of receiving to sending separation is computed based on the samples obtained in the epoch (line 12). A long term average (*lavg_ratio*) is used to maintain a limited history of ratios and to smooth out the effect of brief network dynamics. It is maintained as an exponential running average over the last six short term averages (line 13).

When the *lavg_ratio* goes beyond a threshold, the rate is decreased. This threshold, as described earlier, is $1 + \alpha \times$ *receive_packet_sep*. However, rather than using the receive separation of one packet, the average receive separation computed in the last epoch is used, which is obtained as a product of *savg_ratio* and *send_packet_sep*. Hence, if the long-term average ratio is less than the threshold (line 16), then it implies that the network was able to support the previous rate and so the rate control algorithm tries to probe by increasing the rate (line 17). If a flow enters the increase phase, then the decrease variable, δ, is reset to 0.1 (line 18).

If the average ratio is greater than the threshold, then the receiver reduces the rate using the decrease variable δ (line 20). The initial value of δ is 0.1. However, if the average ratio is greater than the threshold for consecutive epochs, it implies that the network resources have decreased or that the flow might have to give some room to another flow. Hence, the rate control algorithm doubles the value of δ, subject to a maximum of 0.5 (line 21).

4.3. Reliability

As described earlier (section 2), RTT estimates are inaccurate primarily due to the low bandwidth nature of the wireless links. Hence, WTCP does not have a notion of a retransmission timer. WTCP depends on the following two techniques for retransmission of packets.

1. *SACK algorithm.* The important steps in processing an acknowledgment are shown in figure 2 as a pseudo-code. In the figure, *p.rel_seq_num* represents the reliability sequence number of the packet and *p.cctrl_seq_num* represents the congestion control sequence number of the packet. For every transmitted packet the sender keeps

```
process_ack(acknowledgment a)
1 foreach packet p in the unacked list
2     if ($p.rel\_seq\_num < a.CACK$) /* The cumulative ack is acking the packet */
3         remove packet from unacked list
4     else if ($p$ is $SACKed$) /* The selective ack is acking the packet */
5         remove packet from unacked list
6     else if ($p.ready\_for\_retx = FALSE$) /* Packet hasn't been marked */
7             and ($p.cctrl\_seq\_num < a.hcctrl\_seq\_num$) /* Ack corresponding to a later packet */
8         $p.ready\_for\_retx \leftarrow TRUE$
```

Figure 2. The reliability algorithm for WTCP at the sender.

track of the congestion control sequence number of the packet which was used to send it. The ACK processing algorithm has a faster implementation but the presentation here is aimed towards simplicity. Lines 2–8 are executed for all unacked packets. If the packet is being CACKed or SACKed, then it is removed from the unacked list (lines 2–5). A packet is marked for retransmission when the sender recognizes that the receiver has seen a packet with a larger congestion control sequence number. Packets marked ready for retransmission have a higher transmission priority compared to the new data packets.

2. *Probe packets.* While the SACK algorithm described above effectively detects "holes", it will not recover the loss of a packet unless another packet that was subsequently transmitted has been successfully received at the receiver. Consequently, it is possible for the sender to lose a sequence of packets, and then run out of more packets to send. Since there is no concept of a retransmission timer, lost packets would never be recovered.
   In order to solve this problem, when the sender does not have any data to send but the rate indicates that a new packet can be sent out, it sends a probe packet instead. The receiver responds to a probe packet with an acknowledgment, and now the *hcctrl_seq_num* field in the ACK contains a congestion control sequence number that is larger than that of any transmitted packet. As a result, the SACK algorithm described above kicks in for the detection and retransmission of packets (figure 2, lines 6–8). Note that probe packets are also sent periodically during the blackout period with increasing congestion control sequence numbers, in order to elicit an acknowledgement from the receiver.

5. Performance evaluation

We currently have a user-level implementation of WTCP that we have evaluated over the CDPD network, and a simulation model of WTCP in the *ns-2* simulator. In this section, we present the results of our performance tests of WTCP through practical experiments and simulations.

5.1. Experiments on the CDPD network

We performed two sets of experiments on the CDPD network accessible in Chicago, Illinois and Bloomington, Illinois. The first set of experiments was aimed towards understanding the characteristics of the CDPD network. These results were then used to simulate a CDPD like environment and obtain performance comparisons with other TCP approaches. The second set of experiments compared an off-the-shelf implementation of TCP Reno available with Windows 95C and Linux 2.0.33 against a user level WTCP implementation on the same platforms. For all our experiments, the sender was a 233 MHz Pentium PC running Linux 2.0.33, which was used as a fixed host. The receiver was a 233 MHz Pentium Winbook laptop running Windows 95C and was used as the mobile host. The CDPD device was a Sierra Wireless AirCard.

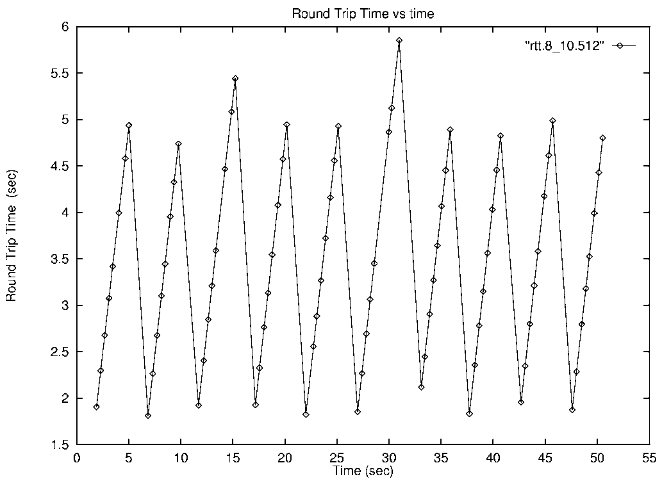
We performed CDPD experiments at different locations, and speeds of 0–55 mph at different times of the day. Due to space constraints, we only present typical results obtained while traveling at 55 mph.

5.1.1. CDPD network characteristics

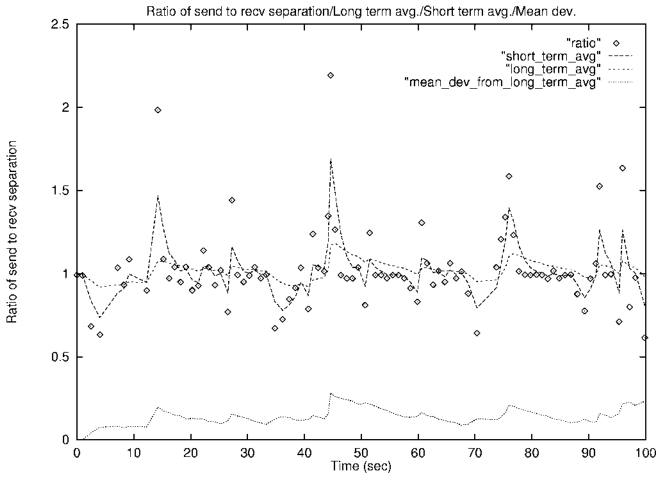
To find out the random loss percentage while moving at 55 mph, we sent UDP packets of 512 bytes at 1 s intervals over 30 min. The short term observed error rate varied between 1–15% with a long term average of around 4%.

For studying the effect of back to back packets on round trip time, we performed several experiments with varying burst sizes. The RTT increases linearly within a burst. The typical values of RTT observed for a packet size of 512 bytes is about 1.8 s. Figure 3(a) shows results from a experiment with 512 byte packets and a burst size of 8. This shows that bursting data packets would result in high RTT values with high variance, resulting in high RTOs. This affects the performance of TCP as is found in TCP experiments, described in section 5.1.2.

For analyzing the effectiveness of the ratio of sender side separation to receiver side separation, we measured the short term average, long term average and the deviation of the ratio. A sample plot for 100 packets sent at an interval of 1 s is shown in figure 3(b). The graph shows the ratio of sender side separation to receiver side separation, the short term average (50% weight on a new sample), the long term average (12.5% weight on a new sample), and the mean deviation of the long term average (12.5% weight on a new sample). The graph clearly shows that the short term average is able to capture current network behavior, and the long term average does not vary so much as shown by the low mean deviation of the long term average.

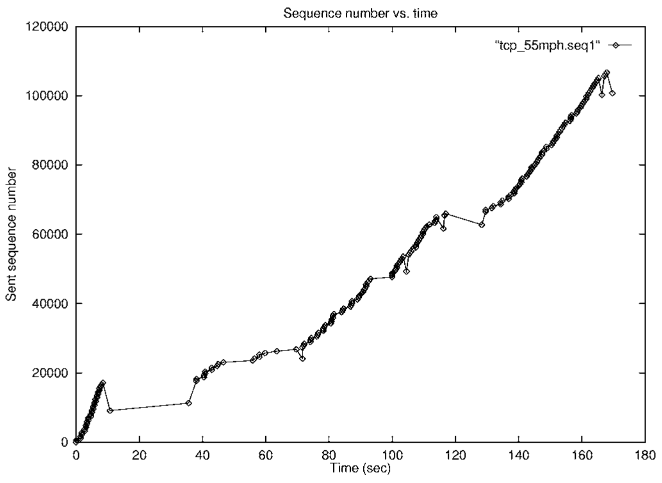


(a)

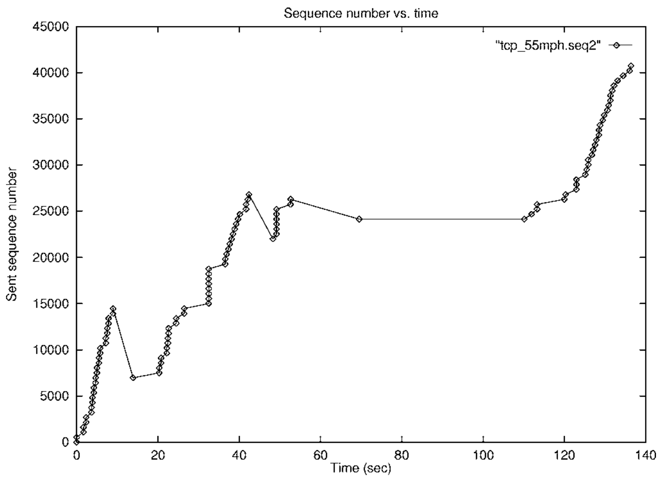


(b)

Figure 3. (a) RTT for burst size of 8 with packet size of 512 bytes. (b) Ratio of send to recv separation for packet sizes of 512 bytes sent at 1 s interval. Also shown are the short term average, long term average and the mean deviation for the long term average.



(a)



(b)

Figure 4. Performance of TCP while moving at 55 mph. A blackout period is captured in (b).

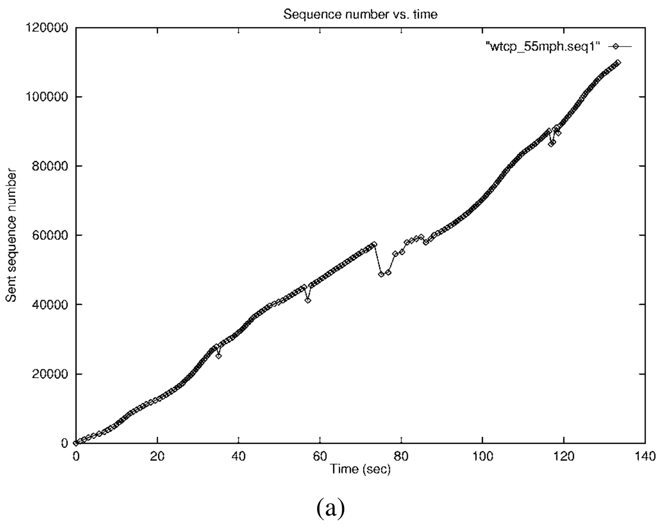
5.1.2. TCP versus WTCP

We sent 100 KB of data using TCP Reno and WTCP across various experimental conditions. At an average TCP takes 134 s to send 100 KB of data from the fixed host to the mobile. WTCP takes less than 100 s on average for sending the same amount of data. The total time taken by WTCP does not vary with varying loss percentage, as the sending rate in WTCP is not affected by random losses. In figures 4 and 5 we present two typical TCP and WTCP performance results, which were obtained while moving at a speed of 55 mph.

In figure 4(a), there are 8 random losses perceived at the sender. TCP stalls at periods as it is waiting for reception of acks to move/increase its congestion window. There are two retransmission timeouts, one at 35 s (RTO 30 s) and another at 128 s (RTO 17 s). For all other losses, TCP is able to recover using fast retransmit. The large retransmission timeout at approximately 35 s is because of the slow start mechanism which aggressively fills up the buffers, resulting in large RTTs and correspondingly large RTOs. In figure 4(b), there are two sequences of losses one from 13 to 26 s and the other one from 42 s to 120 s. During these blackout periods, TCP reacts using the RTO mechanism and even backs off its RTO value resulting in poor performance. It is doing slow start in the beginning and after every retransmission timer run out. These slow starts further skew the RTT and RTO computations because of a fast queue buildup, as discussed earlier.

In figure 5(a), there are 11 losses. At around 70 s, WTCP enters into blackout and stops transmission for a few seconds, till it starts getting acknowledgments from the receiver. But overall, the rate of transmission is not effected by the random losses. In figure 5(b), there are 3 random losses and a blackout period of about 60 s. As soon as the blackout period is over, WTCP starts sending at the rate it was using before going into blackout.

A comparison of figures 4(a) and 5(a) shows that for approximately the same number of losses, TCP takes 170 s for sending 100 KB data whereas WTCP takes 130 s (inclusive of a short blackout). Similarly for Figures 4(b) and 5(b), we



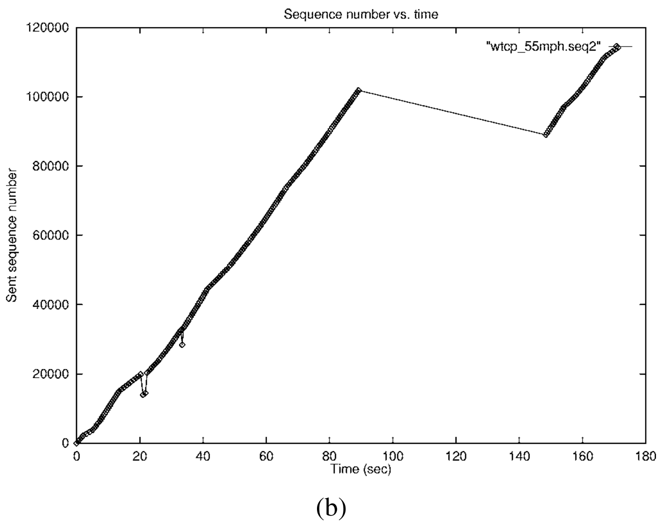


Figure 5. Performance of WTCP while moving at 55 mph. A blackout period was captured in (b).

find that TCP takes 140 s to send 40 KB data, whereas in the same amount of time, WTCP is able to send about 80 KB of data. As TCP does not distinguish between congestion and random losses and has no mechanism to handle blackouts, these results were expected ofTCP. Also, as WTCP has smarts to distinguish between congestion and random losses, and also recognizes blackouts, expectedly, it performs much better than TCP on the real CDPD network.

5.2. Simulations

In this section we compare the performance of WTCP, TCP-NewReno, TCP-Vegas and Snoop-TCP via simulations using the *ns-2* simulator. The WTCP protocol has been implemented in *ns-2* as a WTCP-Agent and WTCP-sink pair. The WTCP-sink computes the new value for the send packet separation and this information is carried back to the sender with every ack. The network topology used in the simulations is shown in figure 6. As we can see from the figure, the packet transmission latency over the wireless network will dominate the latency over the wireline part of the connection. Thus, the results for this topology are representative of networks with more complex wireline topologies. Even though this topology does not mimic all the characteristics of any specific wireless wide area network, it gives a clear picture of the performance of various transport protocols over a channel with low bandwidth and significant error rates. While networks such as CDPD currently offer only 19.2 Kbps raw bandwidth, it is expected to grow by two or three times in the future [23]. So results have been presented using 20 Kbps and 50 Kbps bandwidth for the wireless channel. Also, the observed error rate varies widely depending on factors such as the location of the mobile host and its traveling speed etc. So for the simulations we have used an exponential error model with mean error rates ranging from three to eight percent.

The rest of this section presents four different sets of results. For the first set of results we have used a single flow to evaluate the performance of TCP-NewReno, TCP-Vegas and

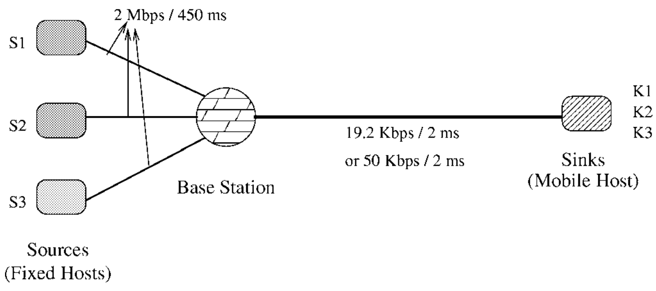


Figure 6. Network topology.

WTCP with different bandwidths and error rates. As congestion control is primarily concerned with the total number of packets successfully sent and not the specific sequence number of the packet, we use the total number of packets received at the sink to compare their performances. The next subsection uses multiple flows to study the aggregate throughput obtained and the fairness characteristics of WTCP. In the third subsection we compare WTCP with TCP-Reno using the Snoop protocol. As the Snoop protocol in the *ns* simulator as of version 2.1b4 is broken, we implemented the Snoop as a Queue object that buffers packets being transmitted to the mobile host, filters duplicate acknowledgments if necessary and retransmits buffered packets on the arrival of the very first duplicate ack. This reflects Snoop behavior accurately. Sufficient buffer space was provided to avoid overflow. All the simulations presented in this section use a packet size of 500 bytes.

5.2.1. Single flow

The results for this scenario shown in figures 7, 8 and 9, use a wireless bandwidth of 50 Kbps, a queue size of 5 packets and error rates of 0%, 4% and 6%, respectively. Figure 7 demonstrates that WTCP provides the same throughput as TCP-Vegas when the channel is error free. The next two figures show that the performance of NewReno and Vegas degrade significantly with increase in error rate. The figures indicate that for a 50 Kbps channel with an error rate of 4%, WTCP

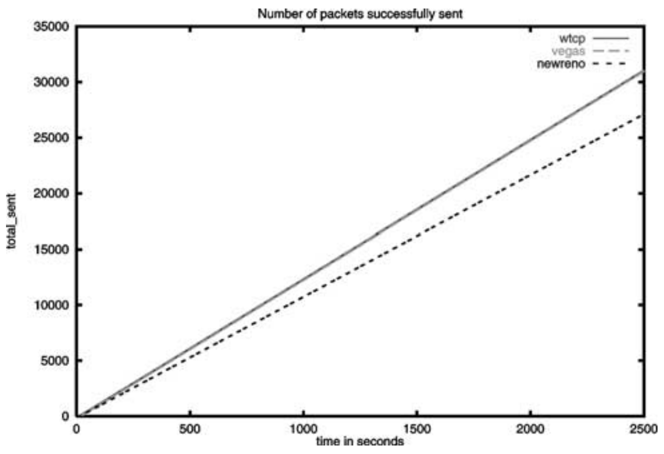


Figure 7. 0% error, 50 Kbps.

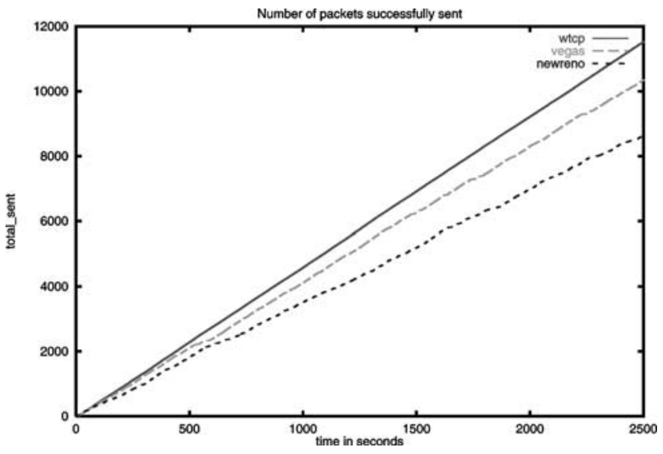


Figure 10. 4% error, 20 Kbps.

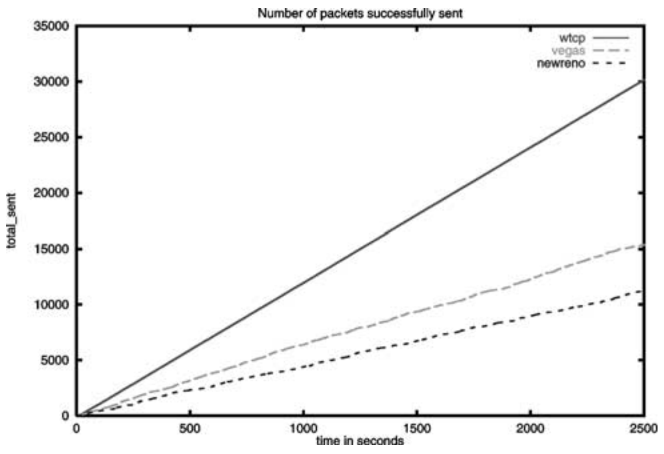


Figure 8. 4% error, 50 Kbps.

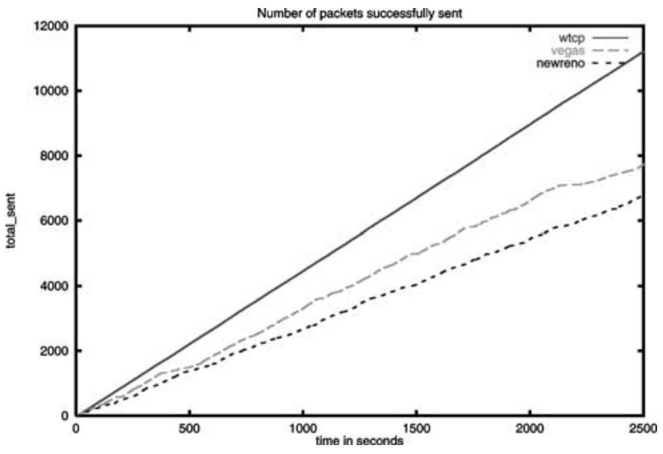


Figure 11. 6% error, 20 Kbps.

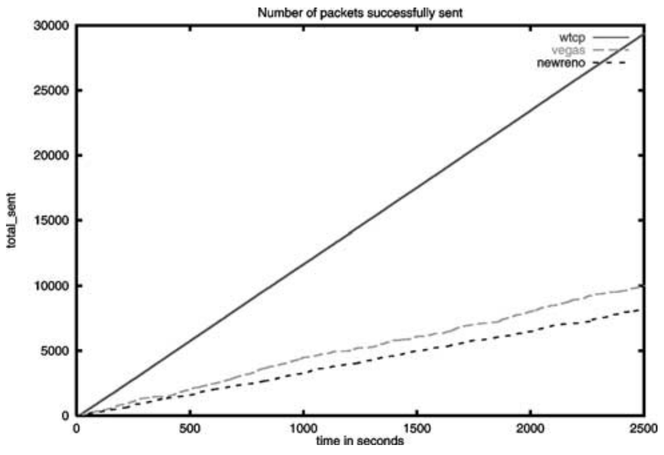


Figure 9. 6% error, 50 Kbps.

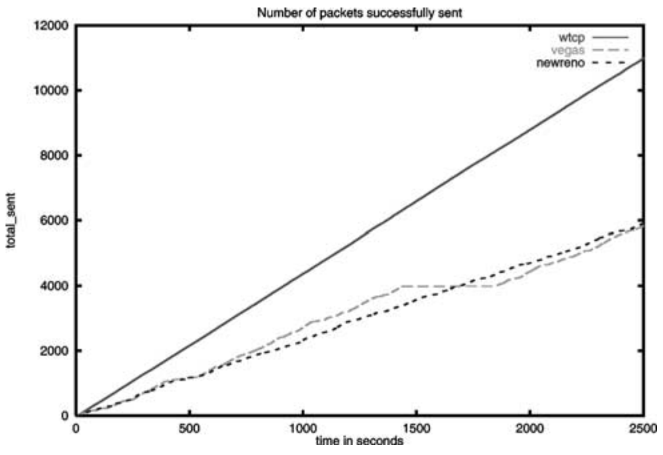


Figure 12. 8% error, 20 Kbps.

provides about 100% improvement in performance over the other versions of TCP we have considered. Figures 10, 11 and 12 use a wireless bandwidth of 20 Kbps and error rates of 4%, 6% and 8%, respectively. The difference in performance between WTCP and other mechanisms are less pronounced with a smaller bandwidth because for a single flow, the packets already in the queue keep the channel occupied for a larger period of time, thereby reducing the impact of the drastic reduction of the sender's congestion window. Whereas with multiple flows, the buffer space available for a single flow will be less, and hence, the resultant throughput will decrease. The results in this section indicate that in wide area wireless networks, WTCP clearly outperforms its wire-line end-to-end counterparts.

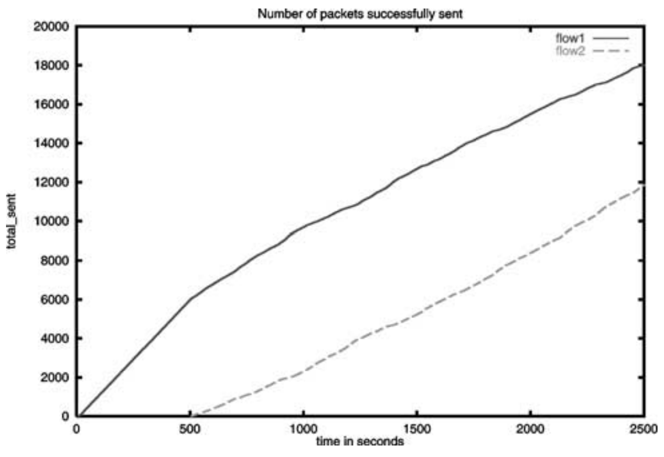


Figure 13. New flow.

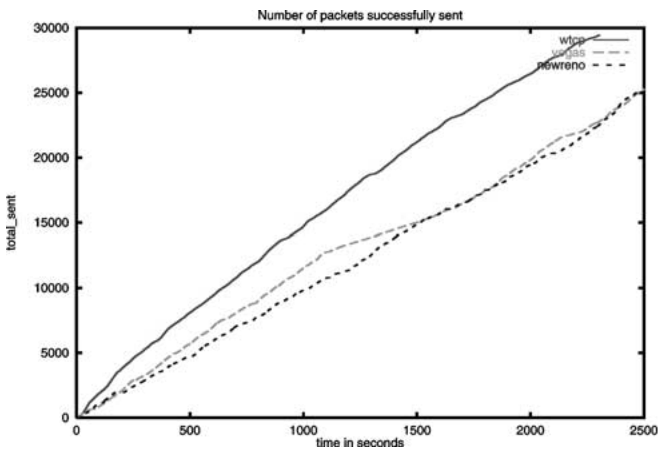


Figure 14. Aggregate sent.

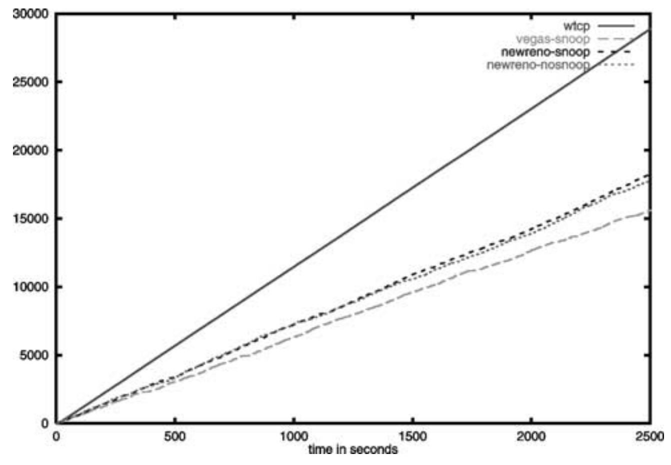


Figure 15. Snoop: 50 Kbps, 4% error.

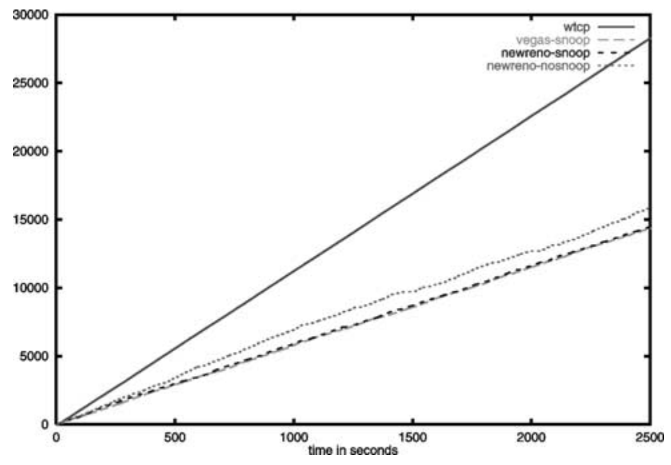


Figure 16. Snoop: 50 Kbps, 6% error.

5.2.2. Multiple flows

In this subsection, we study the behavior of WTCP with multiple flows. For all the cases considered we have used a queue size of 50 packets and bandwidth of 50 Kbps and an error rate of 4% for the wireless channel. In the scenario shown in figure 13, a new flow is introduced into the system when another existing flow is in a steady state. The introduction of a new flow results in an increase in the receive packet separation for the old flow. As a result, the old flow increases its send packet separation thereby providing more room for the new flow. The figure indicates that the new flow manages to obtain the fair share of the wireless bandwidth in a short period of time. The parallel lines clearly show that the wireless bandwidth is equally divided between the new and the old flows. Figure 14 gives the aggregate throughput obtained by three flows started at the same time. The aggregate throughput obtained by all the three flows with WTCP is higher than that obtained with the other mechanisms. To measure the fairness, we also used a fairness index [9] given by

$$F(x) = \frac{\left(\sum x_i\right)^2}{n\left(\sum x_i{}^2\right)},$$

where x_i is the share of bandwidth obtained by a flow i and n is the total number of flows. The fairness index is a continuous function in [0, 1], and the larger the index value, the better the fairness. All the three congestion control mechanisms considered were fair with the indices ranging between 0.978 and 1.0. This set of simulations also serves to indicate that irrespective of the starting point of the flows, they all converge to fairness.

5.2.3. Snoop protocol

Figures 15 and 16 correspond to the scenario in which a 50 Kbps wireless channel was used with the snoop protocol being run at the base station. The queue size was set to 5 and the error rates were 4% and 6%, respectively. Also the channel between the source and the base station was 1 Mbps with a latency of 50 ms and the latency between the base station and the sink being 200 ms. Since the latency between the Snoop agent and the mobile host dominates the round trip time, the use of Snoop does not translate into an improvement in performance. Also the performance of TCP-Vegas deteriorates because Snoop interferes with the early congestion adaptation mechanisms. Figure 16 considers the case where the error rate is 6%. In this case too, WTCP performs better than TCP-NewReno using Snoop. TCP-NewReno reacts drastically to self-induced congestion losses. WTCP, however, uses the decrease variable which provides a graded reaction. Note that the use of Snoop cannot improve the performance of TCP-NewReno beyond the case where there are no losses caused by channel errors. As we have seen in all the subsections above, even with losses, WTCP performs better than TCP-NewReno. Snoop is quite effective in hiding the losses in the wireless channel from the source if the delay experienced after the point where the Snoop protocol executes is very small compared to the total round trip delay. In other cases, as seen from figures 15 and 16, it does not provide any improvement.

6. Other related work

One of the key features of WTCP is its ability to detect wireless losses. Kunniyur and Srikant show in [15] that for loss based LIMD (Linear Increase Multiplicative Decrease) algorithms to work well in wireless networks, the sender must be able to exactly distinguish between random losses and congestion losses. Biaz and Vaidya show in [5] that, unfortunately, most currently proposed "loss predictors" (i.e. metrics that are used to distinguish random losses from congestion losses) such as those based on the congestion detection metrics in [6], do not work well in practice. These results further motivate our approach of not using packet losses as the primary metric for congestion detection. In WTCP we use the ratio of the receiver-to-sender inter-packet separation in order to determine whether to initiate the increase phase or the decrease phase of the LIMD algorithm. Biaz and Vaidya also use this metric in [5], but only to predict if the loss is congestion-related or not. We believe that detecting and reacting to incipient congestion is one of the powerful mechanisms in our approach. Even in wireline networks, algorithms such as TCP Vegas have tried to detect and react to incipient congestion and reduce or eliminate packet loss [6]. However, the specific mechanisms of TCP Vegas are prone to unfairness and are not applicable for low bandwidth wireless networks (because round trip time is highly affected by packet bursts in TCP Vegas), as shown in this paper.

Some wireless TCP approaches try to use the existing features of TCP to take care of mobility and high error rates. Cáceres and Iftode [8] have proposed a mechanism based on fast retransmits. After the mobile registers with a new base station, it enters into the fast retransmit mode and also sends a signal to the other end to do the same. One of the ways this signaling can be done is by sending three duplicate ACKs. Similarly, the M-TCP [7] approach, which is based on the split connection approach, makes use of the persist mode of TCP. At the split point of the TCP connection (or proxy), an ACK for all but the last byte is forwarded to the fixed host. On detection of a link failure, the TCP layer at the mobile is frozen. The proxy on receiving no ACKs from the mobile, advertises zero window along with the ACK for the last byte, thus putting the fixed host in persist mode. When the link is up, the proxy receives an ACK from the mobile, resulting in the proxy informing the fixed host of the true window and thus restarting TCP.

There are several similarities between the space communication environment and the mobile and wireless environments, such as link outage, high latency, varying RTT, data corruption etc. As a result, research on transport layers for satellite networks, is also relevant for our work. Durst et al. [11] propose SCPS-TP, a protocol for space communications, which has several mechanisms for enhancing TCP to counter the problems of data corruption, link asymmetry and limited bandwidth. The mechanisms include using ICMP messages to distinguish various losses, header compression, use of an efficient selective negative acknowledgment (SNACK) scheme, etc. Henderson and Katz [12] have proposed STP as a transport layer protocol for use over a satellite link to a mobile or over a link connecting two satellites. STP has very low reverse traffic as it is not clocked by ACKs from the receiver. Instead, the sender polls periodically to enquire about the status of the receiver's buffer. The receiver can also send an unsolicited status update message to the sender.

Most of the above solutions are designed for taking care of one or more general wireless problems such as high bit error rate, high latency, handoff etc. The congestion control algorithm of all of these solutions is based on that of TCP. Hence, it is not clear how these would perform over WWANs (such as CDPD, RAM, Ardis etc.). In contrast, WTCP addresses issues specific to WWANs and proposes a rate control algorithm which is fundamentally different from TCP and which works efficiently in WWAN environments. WTCP is also extensible to environments with higher bandwidths and lower delays such as wireless LANs.

7. Issues

In this section, we present some of the design and implementation issues that need to be addressed in WTCP. We also briefly explain proposed future work in WTCP.

- *Loss prediction mechanisms.* As mentioned earlier in the paper, WTCP uses loss predictors that are based on the average packet separation observed at the receiver. Using this predictor, we are able to correctly predict random losses. However, a few congestion losses could be mistakenly predicted as random losses. This results in an undesirable late reaction to congestion losses. Though we observed very few instances of such mispredictions in our tests with WTCP, we are working to improve the efficiency of predictors that determine the nature of packet loss.
- *Epoch based updates.* The current implementation of WTCP does updates at the end of an epoch. Once the receiver sends an update to the sender at the end of the epoch, the receiver does not start counting packets for evaluating averages until the sender has confirmed receiving this update. Hence, the packets sent in the intervening

period (which is at least one round trip time) are not utilized for rate control.

- *Internet behavior.* We have demonstrated the effectiveness of WTCP in networks where the wireless latency dominates the end-to-end delay and the variations in the delays experienced due to queueing at the wireline switches do not have a significant impact on the end-to-end delay. We are currently investigating the performance of WTCP on wireless networks with varying link delay.
- *Impact of medium access protocol.* For the common case of data transfer from the base station to the mobile host, the base station typically controls access to the wireless channel. However, for the uplink case, the MAC protocol determines which host obtains access to the channel, and in such a situation, contention among hosts could disturb the inter-packet separation of flows. For example, with CSMA/CA, the host that gets the channel has a greater probability of channel access for the next time slot. Thus, uplink flows may not be rate based. This affects the rate control mechanism of WTCP.
- *Buffer overflow.* Since WTCP does not clock data packets and is rate-based, the sender does not stop transmitting packets unless the receiver instructs it to do so or a blackout is detected. The timeout period for declaring a blackout is typically quite large (on the order of a few RTTs). Hence, there is a possibility of a buffer overflow at the sender before detection of a blackout.
- *Granularity of inter-packet separation.* Wireless LANs and MANs have bandwidths that are an order of magnitude higher than WWANs. Thus, the inter-packet separation for flows in such networks is bound to be much smaller than that for WWANs. This decreases the threshold for increase/decrease phase of flows. As a result, we have observed that for bandwidths greater than 2 Mbps, the efficiency of WTCP drops to around 65%. We are looking at ways to improve WTCP's efficiency on higher bandwidth networks.
- *Multi-hop radio networks.* While WTCP has been targeted at last-hop wireless wide-area networks, we think it will apply equally well in multi-hop wireless networks, such as Metricom [18]. In addition to considering such networks as part of future work in this area, we are also investigating WTCP in the context of ad hoc networks, which are multi-hop wireless networks where all the links are wireless and all nodes are mobile.

8. Conclusion

WTCP is rate-based and the rate adjustment is performed at the receiver; consequently, WTCP does not burst packets, overcomes the problems of inaccurate round trip time computations, and handles asymmetric channels. WTCP uses inter-packet separation at the receiver as the primary metric for rate control with congestion-related loss detection as the backup mechanism; responding early to incipient congestion helps to keep the algorithm stable and at the same time, handling congestion-related losses causes WTCP to react correctly and fairly in worst-case scenarios of sudden congestion peaks. WTCP uses SACK and no retransmission timers for loss recovery; this enables efficient loss recovery without going into prolonged timeouts in the worst case. WTCP also provides for recovery from blackouts and good startup behavior for short-lived flows. In summary, WTCP handles most of the problems of WWAN, including those traditionally ignored by related work such as large and varying round trip times, and the significant fraction of the delay being incurred in the wireless segment.

References

[1] J. Agosta and T. Russle, *CDPD: Cellular Digital Packet Data Standards and Technology* (McGraw-Hill, New York, 1997).

[2] A. Bakre and B.R. Badrinath, I-TCP: Indirect TCP for mobile hosts, in: *Proceedings of International Conference on Distributed Computing Systems* (1995).

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

[4] H. Balakrishnan, S. Seshan, E. Amir and R. Katz, Improving TCP/IP performance over wireless networks, in: *Proceedings of ACM MOBICOM* (1995).

[5] S. Biaz and N.H. Vaidya, Discriminating congestion losses from wireless losses using inter-arrival times at the receiver, in: *Proceedings of IEEE Application Specific Systems and Software Engineering Technology* (1999).

[6] L. Brakmo and L. Peterson, End-to-end congestion avoidance on a global internet, IEEE Journal on Selected Areas in Communications 13(8) (October 1995) 1465–1480.

[7] K. Brown and S. Singh, M-TCP: TCP for mobile cellular networks, ACM Computer Communications Review 27 (October 1997) 19–43.

[8] R. Cáceres and L. Iftode, Improving the performance of reliable transport protocols in mobile computing environments, IEEE Journal on Selected Areas in Communications 13(5) (June 1995) 850–857.

[9] D. Chiu and R. Jain, Analysis of the increase/decrease algorithms for congestion avoidance in computer networks, Journal of Computer Networks and ISDN 17(1) (June 1989) 1–14.

[10] A. DeSimone, M. Chuah and O. Yue, Throughput performance of transport-layer protocols over wireless LANs, in: *Proceedings of IEEE GLOBECOMM* (1993).

[11] R. Durst, E. Travis and G. Miller, TCP extensions for space communications, Wireless Networks 3(5) (1997) 389–403.

[12] T.R. Henderson and R.H. Katz, Transport protocols for Internet-compatible satellite networks, IEEE Journal on Selected Areas in Communications 17(2) (February 1999) 345–359.

[13] S. Keshav, Congestion control in computer networks, PhD thesis, UC Berkeley (1991).

[14] T. Kim, S. Lu and V. Bharghavan, Improving congestion control performance through loss differentiation, in: *Proceedings of IEEE International Conference on Computers and Communication* (1999).

[15] S. Kunniyur and R. Srikant, Fairness of congestion avoidance schemes in heterogeneous networks, in: *Proceedings of International Teletraffic Congress* (1999).

[16] T.V. Lakshman and U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM Transactions on Networking 5(3) (June 1997) 336–350.

[17] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, TCP selective acknowledgement options, Internet RFC 2018 (October 1996).

[18] Metricom, `http://www.metricom.com/`

[19] T. Nandagopal, T. Kim, P. Sinha and V. Bharghavan, Service differentiation through end-to-end rate control in wireless packet networks, in: *Proceedings of IEEE Mobile Multimedia Conference* (1999).
[20] V. Paxson, On calibrating measurements of packet transit times, ACM SIGMETRICS Performance Evaluation Review 26(1) (June 1998) 11–21.
[21] S. Shenker, Some conjectures on the behavior of acknowledgment-based transmission control of random access communication channels, in: *Proceedings of ACM Sigmetrics* (1987).
[22] The Qualcomm High Data Rate Wireless Network, `http://www.qualcomm.com/hdr/`
[23] Wireless Data Forum, `http://www.wirelessdata.org/`
[24] R. Yavatkar and N. Bhagawat, Improving end-to-end performance of TCP over mobile internetworks, in: *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications* (1994).

Prasun Sinha received his B.Tech. in computer science and engineering from IIT Delhi, India, in 1995, his MS in computer science from Michigan State University in 1997 and his PhD in computer science from University of Illinois at Urbana-Champaign in 2001. He is currently with Bell Labs, Lucent Technologies. His research interests are in computer networking and mobile computing.
E-mail: prasun@dnrc.bell-labs.com

Thyagarajan Nandagopal is a PhD candidate in the Electrical and Computer Engineering Department at the University of Illinois and is affiliated with the TIMELY Research Group. His research focuses on providing Quality-of-Service in wireless networks.
E-mail: thyagu@timely.crhc.uiuc.edu

Narayanan Venkitaraman is currently with Motorola Labs., Schaumburg. He received the B.E. degree in computer science and engineering from Anna University, India, in 1997 and the M.S. degree in computer science from University of Illinois at Urbana-Champaign in 1999.
E-mail: venkitar@labs.mot.com

Raghupathy Sivakumar received the B.E. degree in computer science and engineering from Anna University, India, in 1996 and the M.S. and PhD degrees from the University of Illinois at Urbana-Champaign in 1998 and 2000, respectively. He is currently an Assistant Professor in the Electrical and Computer Engineering Department at Georgia Institute of Technology. His research interests are in wireless networks and mobile computing, network Quality-of-Service, and programmable networks.
E-mail: siva@ece.gatech.edu

Vaduvur Bharghavan is an Associate Professor in the Electrical and Computer Engineering Department at the University of Illinois, where he heads the TIMELY Research Group. His research interests are in mobile computing and computer networking.
E-mail: bharghav@timely.crhc.uiuc.edu