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

Abstract

Wireless wide-area networks (WWANs) are characterized by very low and variable bandwidths, very high and variable delays, significant non-congestion related loss, 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 is designed to operate efficiently and fairly 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 initial results indicate the 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.

1 Introduction

Recent years have witnessed an explosive growth in the use of wireless wide-area networks (WWANs) such as CDPD, RAM and Ardis [1], with industry projections topping $2.5 billion by the year 2002. Many large corporations are equipping their mobile workforce 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.

Despite the enormous commercial interest, a typical connection over a WWAN network observes very low and highly variable throughput (between $O(100bps) - O(10Kbps)$), very high and highly variable latency (between $O(400ms) - O(4sec)$), bursty and random packet losses unrelated to congestion (1% - 10%), and occasional blackouts exceeding 10 seconds. Under such operating conditions, standard transport protocols such as TCP perform very poorly 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 ($RTO = (\overline{rtt}) + 2 \times \Delta(rtt)$). In WWAN networks, 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 seconds 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 [2], (b) providing TCP-aware smarts in the base station[3], and (c) splitting the TCP connection into two parts [4, 5] - one in the backbone network and one over the wireless link running a specialized protocol [5]. 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

[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 transfered 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.

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, 3, 4, 5]. We choose end-to-end mechanisms for three reasons. First, we believe that end-to-end mechanisms are necessary to effectively address the problems of WWAN environments. 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. 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 [6]. 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. 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 currently have a user-level implementation of WTCP that we have evaluated over the CDPD network, and a simulation of WTCP in the ns-2 simulator. Initial performance results indicate that WTCP significantly improves on the state-of-the-art solutions for wireless transport.

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 implementation over CDPD and simulation in ns-2. Section 6 compares WTCP to related work. Section 7 concludes this paper.

2 Characteristics of WWAN Environments

At a high level, most WWAN environments exhibit similar characteristics: the bandwidth is very low and varying, the latency is very high and varying, blackouts exceeding 10 seconds 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. 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 19.2Kbps. Up to 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 12Kbps, and the majority ($\geq 75\%$) 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 [3].

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 - 55mph), location of the user vis-a-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 19.2Kbps. For RAM, the channel is 8Kbps, while for Ardis, the channel may be 4.8Kbps or 19.2Kbps.

   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 800ms to 4 sec. A large fraction of this time is due to transmission on the wireless link (e.g. transmitting a 512 byte packet at 12Kbps takes 300ms), and over 75% of the latency is typically incurred in the mobile switching station$\leftrightarrow$mobile host segment of the connection. In Figure 3(a) we observe the variation in round trip time for UDP handshakes that progress in bursts of 8 from 1.8 sec to 6 sec 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 twice the mean deviation of the round-trip time ($RTO \leftarrow \overline{rtt} + 2 \times \Delta(rtt)$). This may result in very large RTOs (e.g. 32 sec) in CDPD because of two reasons: (a) the 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 in CDPD.

4. *Asymmetric Channel - Bunching of ACKs*: CDPD uses DSMA/CD [7] 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 [8], 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 seconds or more, and results in the back-to-back loss of a sequence of packets. Traveling at 55mph, we observed several blackouts ranging from 10 seconds to 10 minutes 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 inter-packet 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 second. Clearly, interpacket separation is a useful metric to pace the progress of the connection when the network is uncongested or lightly congested. If there are sudden peaks of congestion, this metric becomes less effective as there are fewer samples available for computing the running short-term average of interpacket delay, and must be supplemented with a loss-based metric.

   It turns out that average interpacket delay can also be used as a metric to predict incipient congestion in wireline networks, though it is less useful because the transmission delays are not as significant in higher bandwidth networks. TCP-Vegas uses a variant of this approach by monitoring round-trip times at the sender instead of interpacket delays at the receiver [9]. Of course, this approach 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 Trade-offs in WWAN Environments

Before going into the details of the WTCP design, we need to step back and discuss the trade-offs 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 [2], TCP-aware 'snoop' mechanisms [3], or splitting the connection into two (wireline and wireless) distinct components [4, 5].

Link layer retransmission [2] works well when the latency over the wireless link is small compared to the coarse grain 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 and keeping TCP unchanged at the end hosts simply will not work because link-layer retransmissions will interfere with the reliability and congestion control mechanisms of TCP.

The Snoop protocol [3] 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. As described in Section 2, this approach also assumes that the transmission time over the wireless link is significantly smaller than the coarse grain TCP timer and round trip time. In wireless LAN environments, snoop works well. 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. Additionally, snoop does not work well with some flavors of TCP such as TCP-Vegas. 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 [4, 5] 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 several of the issues listed above. 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 the 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 switch station and the mobile host is the dominant component of a large and varying round trip time makes such approaches less effective. The bottomline 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 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 ($O(100ms)$) 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. Specifically, the ratio of the sender's interpacket delay and the receiver's observed average inter-packet delay provides a responsive metric to determine whether the transmission rate is in steady state, can be increased, or should be 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. In WTCP, the receiver maintains a history of packet losses when the network is uncongested, and computes the expected average and deviation in the number of non-congestion-related packet losses over a time window. Based on this information, it predicts if an observed packet loss is congestion related or not. If the loss is predicted to be due to congestion, then the sending rate is throttled down. Otherwise, the sending rate is adjusted according to mechanism 2 above (to deal with the uncongested or incipient congestion case). We typically observe that congestion-related throttling does not occur often since most of the random losses conform to the pattern generated from the history. However, during blackout periods, this mechanism erroneously assigns blackout-related packet losses to congestion. This causes the receiver to throttle down the sending rate (though the ACKs may not get through to the sender if the blackout is two-way). Once the receiver determines that a blackout has occurred (the sender goes into blackout mode as described in point 6 below), it resets the sending rate to the old (pre-congestion) value upon recovery from blackout.

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, which it then sends back 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 [10], 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.

   This mechanism can also be used to recover from blackouts, which occur when the connection may experience back-to-back losses for extended periods of time due to poor channel conditions or lack of available channels. We have also considered an alternative mechanism for blackout recovery, in which the sender starts to 'probe' the receiver (instead of sending packet-pairs) when it does not receive an ACK for a threshold period of time. The difference in the two cases is whether a successful packet handshake after the blackout reverts to the old transmission rate or computes a new transmission rate according to the new channel conditions. A comparison of these schemes is ongoing work.

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 [11] 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 unacked 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 seconds). If the sender does not receive an ACK in the threshold period, it goes into 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.

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 does not receive an ACK for 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,* RTT_{min}, *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 rate (inverse of sending interpacket delay), the desired ACK frequency, packet size, data, and RTT_{min}, the minimum observed rtt over a recent window of packets. RTT_{min} is used by the receiver in its rate increase algorithm. Other parameters in the ACK packet include the desired sending rate.

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 functions of rate control and reliability in WTCP.

4.1 Rate Control

In order to perform rate control, the receiver maintains the following state: (a) long-term (*lavg_ratio*) and short-term running averages (*savg_ratio*) of the ratio of the observed sender rate computed by the receiver to the actual sender rate of the sender, (b) the maximum expected number of random losses in a given window of packets, and (c) a decrease variable (δ) that indicates the percentage decrease in the desired sending rate if the sending rate needs to be throttled.

At any time, the receiver can be in one of three states:

- *increase*: the long-term and short-term ratios of the observed sending rate at the receiver to the actual sending rate at the sender are both above certain thresholds (i.e. (*lavg_ratio* $> \alpha_+$) and (*savg_ratio* $> \beta_+$)).
- *decrease*: the long-term and short-term ratios of the observed sending rate at the receiver to the actual sending rate at the sender are both below certain thresholds (i.e. (*lavg_ratio* $< \alpha_-$) and (*savg_ratio* $< \beta_-$)).
- *maintain*: otherwise.

```
Initialization
δ ← 20                        /* Percentage of decrease on detection of congestion */
lavg_ratio ← 1                /* Long term average of ratio of receiver to sender's rate */
savg_ratio ← 1                /* Short term average of ratio of receiver to sender's rate */
loss_expected ← 5             /* Estimate of random loss percentage */
loss_observed ← 0             /* Observed loss percentage. It is reset every T_loss time units */
current_phase ← increase      /* It takes one of three values: increase, decrease or maintain */

Constants
α+ ← 0.95                     /* For comparing against lavg_ratio in increase phase */
β+ ← 0.95                     /* For comparing against savg_ratio in increase phase */
α- ← 0.9                      /* For comparing against lavg_ratio in decrease phase */
β- ← 0.9                      /* For comparing against savg_ratio in decrease phase */

decrease_sending_rate()
decrease sending rate by δ %
δ ← MAX(2 × δ,50)
current_phase ← decrease

process_packet()
if (loss_observed was reset more than T_loss time units ago)
    reset loss_observed
if (packet contains data and previous data packet was missing)
    update loss_observed
    if (current_phase = maintain)
        update loss_expected
    if (loss_observed > loss_expected) /* Decrease */
        decrease sending rate by 50%
        δ ← 20
        current_phase ← decrease
else if (packet contains data) /* not a probe or an ack */
    update lavg_ratio
    update savg_ratio
    if (sender has started using the latest computed rate)
        if (lavg_ratio ≥ α+ and savg_ratio ≥ β+) /* Increase */
            increase sending rate by 1/RTTmin
            δ ← 20
            current_phase ← increase
        else if (lavg_ratio < α- and savg_ratio < β- ) /* Decrease */
            decrease_sending_rate()
        else /* Maintain */
            set the new rate to be same as the old rate
            δ ← 20
            current_phase ← maintain
    else if (last action was a speedup) /* sender is using old rate. See if decrease is required */
        if (lavg_ratio < α- and savg_ratio < β- ) /* Decrease */
            decrease_sending_rate()
send ack
```

Figure 1: *Rate Control for WTCP at the receiver*

When the receiver receives a data packet, it checks to see if the packet has the next expected congestion control sequence number. If the packet is received in sequence, *lavg_ratio* and *svag_ratio* are updated. Then the receiver transitions to the *increase*, *maintain*, or *decrease* state, depending on the ratios.

The receiver increases the desired sending rate in the increase state and decreases the desired sending rate in the decrease state according to the algorithm described below. For increase, the short term average captures the effect of increase in available bandwidth, but to avoid a false increase, the long term average is also required to be above a certain threshold (α_+). Similarly for decrease, the short term average captures the effect of reduction in available bandwidth, but to avoid a false decrease, the long term average is also required to be lower than a certain threshold (α_-). The combination of both these averages makes the increase as well as the decrease steps judicious, thus leading to a stable *maintain* phase.

The congestion control algorithm at the receiver, is presented as a pseudocode in Figure 1. The initial values of various variables and values of various constants (α_+, α_-, β_+ and β_-) are also mentioned in the same figure. The particular values used were obtained by tuning WTCP for CDPD while experimenting on the CDPD network in Bloomington, Illinois.

The main components of the congestion control algorithm are described below, with references to line numbers from the Figure 1.

1. *RTT computation:* As discussed earlier, retransmission timeouts are not used in WTCP, however a rough estimate of minimum RTT is used for the following purposes:

   (a) *Probing rate estimation:* $1/RTT_{min}$ is used as the probing rate in the linear increase phase of WTCP. This makes WTCP equally aggressive as TCP Reno in its congestion avoidance phase.

   (b) *Probe interval:* When the WTCP sender goes into the blackout mode, it sends probe packets at an interval of twice $2 \times RTT_{min}$.

   When the sender (re)transmits a packet, it records the transmission time of the packet. The first time that the packet is acknowledged, the sender records the round trip time for the packet. From the reliability mechanisms of WTCP, we retransmit a packet only after determining that the last transmission was lost. Thus, our *rtt* estimates are reliable even for retransmitted packets. RTT_{min} is computed as the minimum *rtt* over a moving window, in order to account for dynamic network conditions. The sender notifies the receiver about the RTT_{min} value, which is used in the increase phase of rate control.

2. *Using loss profile:* WTCP keeps track of the random loss percentage which is computed/updated during the *maintain* phase of the algorithm, as in that phase, losses are least likely to be because of congestion. When a non-consecutive (in terms of congestion control sequence number) data packet is received, the observed loss percentage is updated (Line 4) based on the gap in the congestion control sequence numbers at the receiver[2]. If currently the receiver is in the *maintain* phase, then the expected loss percentage is also updated (Lines 5-6).

   If the observed loss exceeds the worst case expected loss, then the receiver predicts that there has been some congestion loss, halves the desired sending rate, and enters the decrease state.

3. *Increase Phase:* If the received packet is the next expected packet according to the congestion control sequence number, the long term and the short term averages of the ratio of receiving to sending rate are compared with the constants α_+ and β_+ respectively. If the receiving rate is greater than the sending rate or slightly less than the sender's rate, WTCP tries to do probing (Lines 15-18), based on the long term and the short term averages of the ratios. If the increase phase is entered, the decrease variable δ is reset to 20.

4. *Decrease Phase:* WTCP decreases the rate by half on detection of a congestion loss. In other cases, when it decides to decrease based on the long term and the short term averages, the decrease variable δ is used. The initial value of δ is 20. As typically most of the decreases are because of probing, a reduction by 20%, improves the channel utilization to 90% (compared to 75% for TCP Reno). To take care of an actual congestion loss, WTCP doubles the decrease variable for consecutive decreases. Thus, it takes at most one extra RTT, compared to TCP Reno, to drop to 50% (decrease by 20%, followed by a decrease by 40%, as δ is doubled after a decrease, is approximately equivalent to a single drop of 50%) for a real congestion.

[2]In WTCP, we implicitly assume that packets are typically not reordered. Thus, if we observe a gap in the received congestion control sequence number, we treat it as a loss. Experimentation has validated this assumption.

```
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 ← TRUE
9  hack_num_recvd ← max(a.ack_seq_num,hack_num_recvd) /* Update the highest ack number received */
```

Figure 2: *The Reliability Algorithm for WTCP at the sender*

The algorithm enters the decrease phase from three different places (Lines 7, 20 and 27). As described earlier, the loss profile can indicate that a loss is because of congestion, in which case the decrease phase is entered (Lines 8-10). If the sender has started using the latest updated rate and if the long term and the short term averages are below the thresholds α_- and β_- respectively, then also the decrease phase is entered (Lines 19-20). But if the receiver detects congestion after an increase was sent to the sender, then the receiver doesn't wait for the sender to start using the new rate, instead it sends a slow down right away(Lines 25-27).

The idea behind comparison of the average ratios with α_- and β_- (Lines 19 and 26) is to enter decrease phase only when WTCP notices that the receiving rate is falling below the sending rate by a significant amount. Since the short term average (receiving rate to sending rate) might be affected due to a transient network behavior, both the long term and the short term averages of the ratios are taken into account.

5. *Maintain Phase:* This phase is entered when the long term and the short term average ratios indicate that no increase or decrease needs to be done (Lines 21-24). The decrease variable δ is reset to 20 and the current_phase is set to *maintain*.

4.2 Reliability

As described earlier (Section 2), RTT estimates are inaccurate primarily due to the low bandwidth nature of the wireless links. Hence, WTCP doesn't 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 pseudocode. For every transmitted packet the sender keeps 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 through 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). *p.rel_seq_num* represents the reliability sequence number of the packet. 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, at the end of a run of packets, or upon not receiving an acknowledgment for a threshold period of time, the sender sends out periodic probe packets with increasing congestion control sequence numbers. 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 (Figure 2, Lines 6-8).

5 Performance Evaluation

5.1 Experiments on the CDPD network

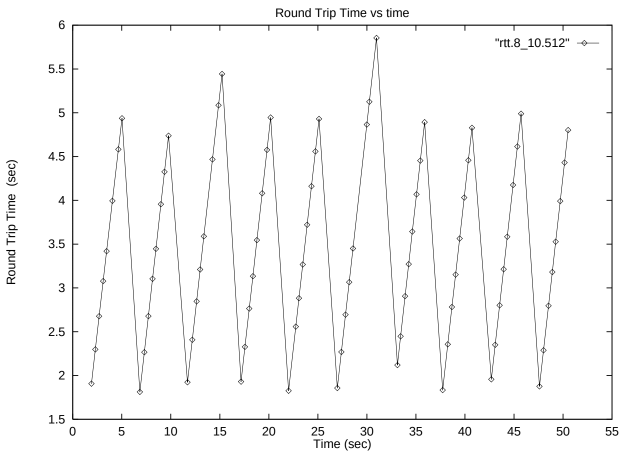
We performed two sets of experiments on the CDPD network accessible in Chicago, Illinois and Bloomington, Illinois. The first set of experiments was aimed towards understanding the characteristics of the CDPD network. These results were then used to simulate a CDPD like environment and obtain performance 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 166 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.

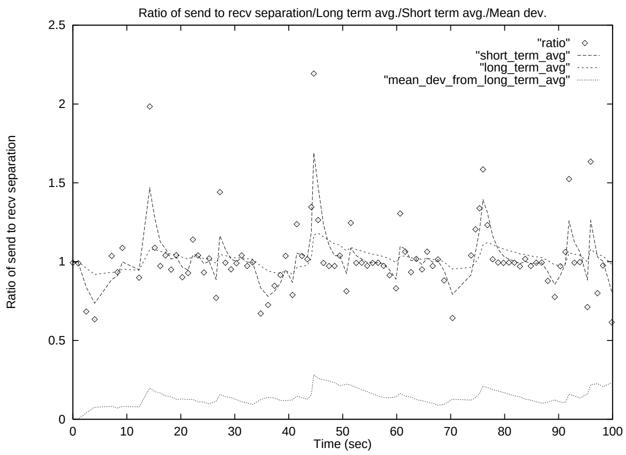
5.1.1 CDPD network characteristics

To find out the random loss percentage while moving at 55mph, we sent UDP packets of 512 bytes at 1 sec. intervals over 30 minutes. The short term observed error rate varied between 1% to 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 sec. 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.



(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 sec. interval. Also shown are the short term average, long term average and the mean deviation for the long term average.

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 sec. is shown in Figure 3(b). The graph shows the ratio of sender side separation to the receiver side separation, the short term average (50% weightage on a new sample), the long term average (12.5% weightage on a new sample), and the mean deviation of the long term average (12.5% weightage 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.

5.1.2 TCP vs. WTCP

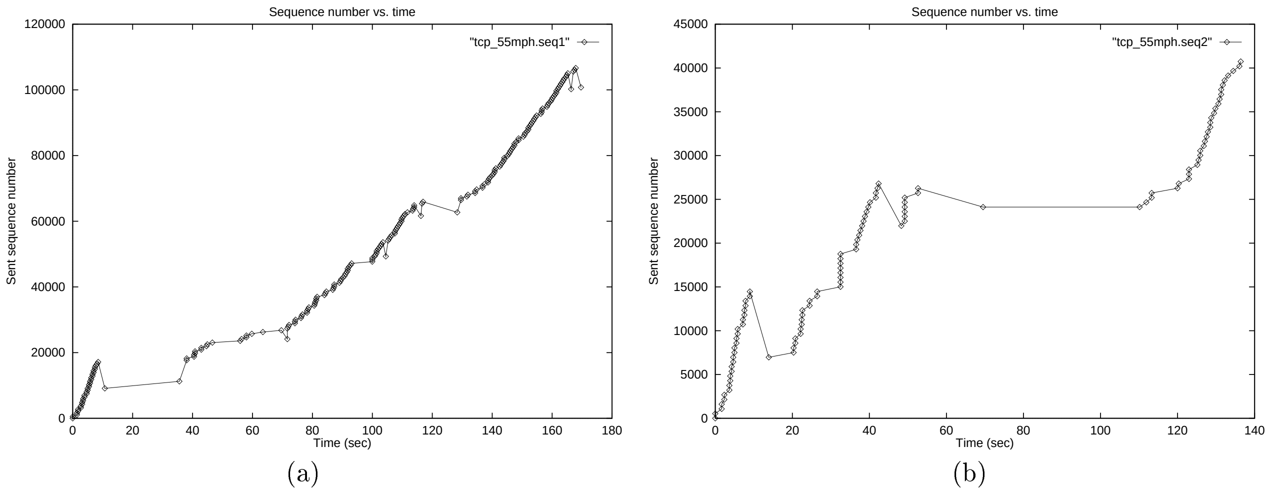


Figure 4: Performance of TCP while moving at 55 mph

We sent 100 KB of data using TCP Reno and WTCP across various experimental conditions. At an average TCP takes 134 seconds to send 100KB of data from the fixed host to the mobile. WTCP takes less than 100 seconds at an 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 TCP and WTCP performance results, which were obtained while moving at a speed of 55 mph.

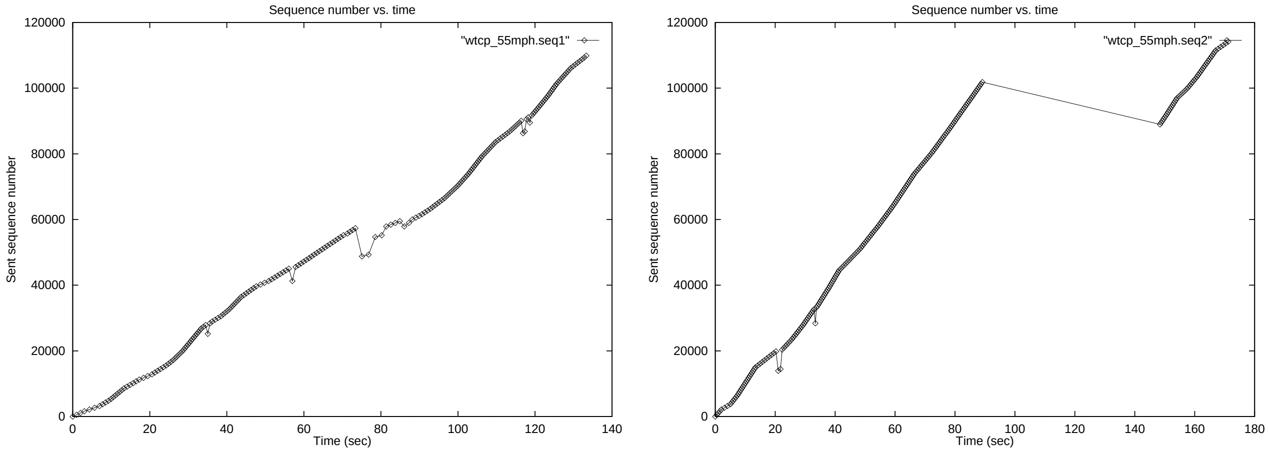
In Figure 4(a), there are 8 random losses perceived at the sender. TCP stalls at periods as it is waiting for reception of acks to move/increase its congestion window. There are two retransmission timeouts, one at 35 sec (RTO 30 sec) and another at 128 sec (RTO 17 sec). For all other losses, TCP is able to recover using fast retransmit. The large retransmission timeout at approximately 35 sec. 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 sec to 26 sec and the other one from 42 sec to 120 sec. 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 build up, as discussed earlier.

In Figure 5 (a), there are 11 losses. At around 70 sec, 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 seconds. 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 seconds for sending 100KB data whereas WTCP takes 130 seconds (inclusive of a short blackout). Similarly for Figures 4(b) and 5(b), we find that TCP takes 140 seconds 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 of TCP. 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



(a) (b)

Figure 5: Performance of WTCP while moving at 55 mph

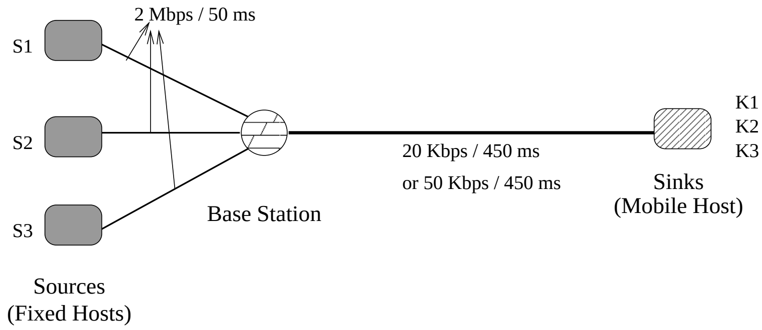


Figure 6: Network topology

sender with every ack. The network topology used in the simulations is shown in Figure 6. 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 19.2Kbps raw bandwidth, it is expected to grow by two or three times in the future [1]. So results have been presented using 20Kbps and 50Kbps 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 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 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 ns 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. Sufficient buffer space we 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 50Kbps, 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 the Newreno and Vegas

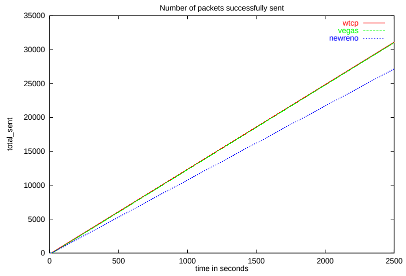


Figure 7: 0% error/50Kbps

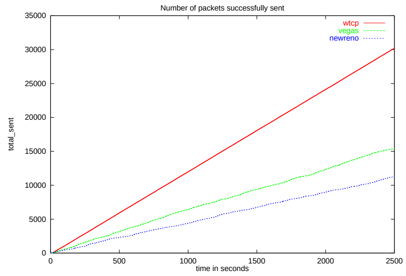


Figure 8: 4% error/50Kbps

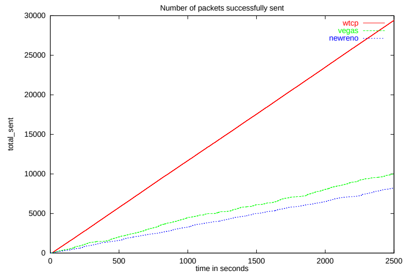


Figure 9: 6% error/50Kbps

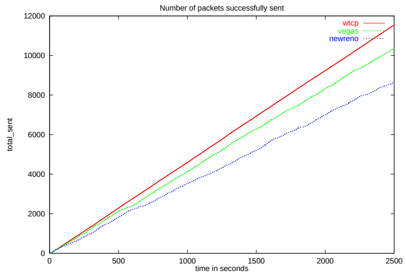


Figure 10: 4% error/20Kbps

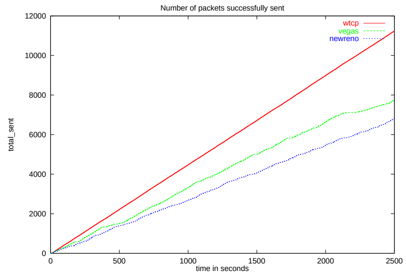


Figure 11: 6% error/20Kbps

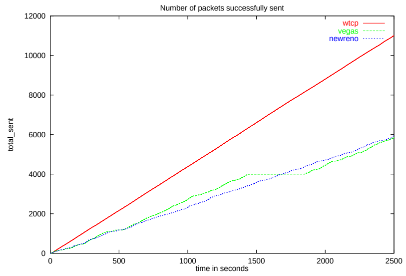


Figure 12: 8% error/20Kbps

degrade significantly with increase in error rates. The figures indicate that for a 50Kbps channel with an error rate of 4%, WTCP 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 20Kbps 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.

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 50Kbps 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 throughputs 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. We also used fairness index [12] given by $F(x) = \frac{\sum {x_i}^2}{n(\sum {x_i}^2)}$, where x_i is the share of bandwidth obtained by a flow i, to measure the fairness. The fairness index is a continuous function in $[0, 1]$, and larger the index value, better the fairness. All the 3 congestion control mechanisms considered were fair with the indices ranging between 0.978 and 1.0. This set of simulations also serve 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 50Kbps 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 1Mbps with a latency of 50ms and the latency between the

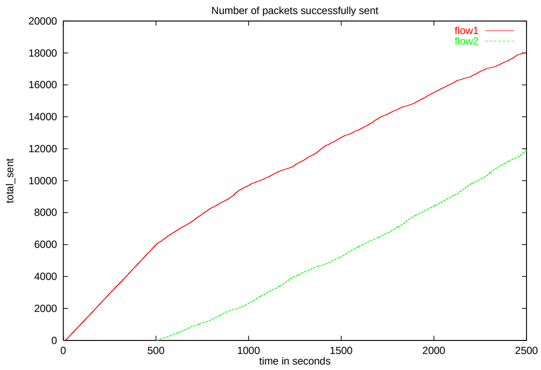


Figure 13: New Flow

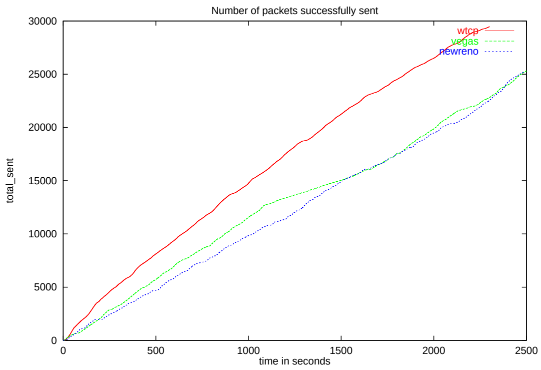


Figure 14: Aggregate sent

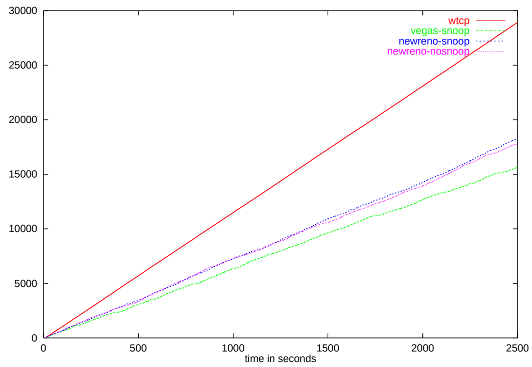


Figure 15: Snoop:50Kbps/4% error

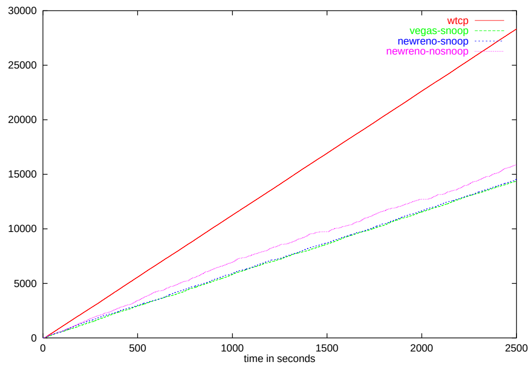


Figure 16: Snoop:50Kbps/6% error

base station and the sink being 200ms. 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 and 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. Though 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 the Figures 15 and 16, it does not provide any improvement.

6 Related Work on Wireless Transport Protocols

The idea of making the link layer more reliable has been investigated by DeSimone et al[2] and also by Balakrishnan [13]. It has been observed that the retransmissions at the link layer may work against the congestion control algorithm at the TCP layer. Since the wireless link has very different characteristics from the wired link, I-TCP [4] and MTCP [5] attempt to separate the congestion control algorithms on the wired and the wireless part using a split connection approach. In the split connection approach, a TCP connection is set up from the mobile to the MSS (Mobility Support Station) and another one from the MSS to the fixed host on the wired network. If the other end is also a mobile, then three individual TCP connections would be required. A specialized transport protocol (such as SRP [5]) for the wireless link can be used, but that would require changes in the TCP layer of the mobile host. Another approach called the snoop-protocol [3] doesn't require any changes in the mobile or the fixed hosts. The snoop module running at the link layer in the MSS inspects TCP headers of packets passing through, and caches the data packets and the ACK information. It retransmits packets either on reception of a duplicate ACK or on expiration of a timer, thereby protecting the fixed host from the wireless losses.

Some wireless TCP approaches try to use the existing features of TCP to take care of mobility and high error rates. Caceres and Iftode [14] 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 [15] 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), 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 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 [16], 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 [17] 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 a 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 sender 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. However, it is not clear how these solutions would perform over WWANs (such as CDPD, RAM, Ardis etc.). This paper proposes WTCP, a transport layer protocol for reliable and sequenced delivery of bytes, for use over WWANs, along with performance comparisons with TCP-NewReno-SACK and TCP-Vegas-SACK using simulations, and with TCP-Reno on a real CDPD network using an implementation of WTCP.

7 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.

We have not addressed several issues in the paper. First, we have not isolated the benefits of each of the mechanisms presented in this paper. Second, we have not compared alternate schemes for tuning ACK frequency and recovering from blackouts. Third, we have not evaluated the performance of WTCP for short-lived flows and more complex scenarios in detail. Fourth, we have not studied the effect of having deep buffers on performance in detail. Finally, we have not explored all the implications of our mechanisms on fairness. These issues, along with a more robust publicly available user-level implementation of WTCP, are the foci of our ongoing work.

References

[1] Wireless Data Forum. http://www.wirelessdata.org/.

[2] A. DeSimone, M.C. Chuah, and O. Yue. Throughput performance of transport-layer protocols over wireless LANs. In *Proceedings of IEEE GLOBECOMM*, 1993.

[3] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP performance over wireless networks. In *ACM MOBICOM*, November 1995.

[4] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proc. 15th International Conf. on Distributed Computing Systems (ICDCS)*, May 1995.

[5] R. Yavatkar and N. Bhagawat. Improving end-to-end performance of tcp over mobile internetworks. In *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.

[6] T. V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. Networking*, June 1997.

[7] John Agosta and Travis Russle. CDPD: Cellular Digital Packet Data standards and technology. McGraw Hill, New York, NY, 1997.

[8] S. Shenker. Some conjectures on the behavior of acknowledgment-based transmission control of random access communication channels. In *Proc. of ACM Sigmetrics '87*, 1987.

[9] L. Brakmo and L. Peterson. End to end congestion avoidance on a global internet. *IEEE JSAC*, 13(8), October 1995.

[10] S. Keshav. *Congestion Control in Computer Networks*. PhD thesis, UC Berkeley, September 1991.

[11] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. Internet Request for Comments 2018 (rfc2018.txt), October 1996.

[12] 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):1–14, June 1989.

[13] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *ACM SIGCOMM*, August 1996.

[14] R. Caceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE JSAC*, 13(5), June 1995.

[15] K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. *ACM Computer Communications Review*, 27(5):19–43, October 1997.

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

[17] Thomas R. Henderson and Randy H. Katz. Transport protocols for internet-compatible satellite networks. *IEEE JSAC*, 1999.