Design and Implementation of a TCP-Friendly Transport Protocol for Ad Hoc Wireless Networks

Zhenghua Fu, Benjamin Greenstein, Xiaoqiao Meng, Songwu Lu
Computer Science Department, University of California, Los Angeles
{zfu,ben,xqmeng,slu}@cs.ucla.edu

Abstract

Transport protocol design for mobile ad hoc networks is challenging because of unique issues, including mobility-induced disconnection, reconnection, and high out-of-order delivery ratios; channel errors; and network congestion. In this work, we describe the design and implementation of a TCP-friendly transport protocol for ad hoc networks. Our key design novelty is to perform multi-metric joint identification for packet and connection behaviors based on end-to-end measurements. Our testbed measurements and ns-2 simulations show a significant performance improvement over standard TCP in ad hoc networks.

1 Introduction

The proliferation of mobile computing devices has spurred interest in the use of mobile ad hoc wireless networks. In such networks, mobile users may exchange data messages and access the Internet at large. TCP has been the predominant transport protocol used in the wired Internet to deliver data; consequently, numerous Internet applications have been developed to run over TCP. It is important to design a transport protocol for mobile users that is both backward compatible and TCP-friendly.

TCP relies on packet loss as an indication of network congestion and triggers efficient congestion control algorithms once congestion is detected. However, it is well-known that TCP is not efficient in ad hoc networks [1][6][7][8]. In addition to congestion, a transport protocol in an ad hoc network must handle mobility-induced disconnection and reconnection, route change-induced packet out-of-order delivery for mobile hosts, and error/contention-prone wireless transmissions[1]. Reaction to these events might require transport control actions different from congestion control. It might be better to periodically probe the network during disconnection than to backoff exponentially [1], and it makes more sense simply to re-transmit a packet lost to random channel error than to multiplicatively decrease the current congestion window [3]. Even if the correct action is executed in response to each type of network event, it is not immediately obvious how to construct an engine that will accurately detect and classify events. Packet loss alone cannot detect and differentiate all these new network events [9].

Much of the literature on TCP-friendly transport for mobile wireless networks endorses a network-oriented approach. In this approach, the network router implements a monitoring mechanism that generates a notification message when it detects an abnormal event so that TCP may react. When mobility triggers network disconnection (called a link failure event), the routing layer sends an explicit link failure notification (ELFN) to the TCP sender [1]; an explicit congestion notification (ECN) message is generated when a congestion loss occurs [5]; and an explicit loss notification (ELN) is sent to the TCP sender when the router observes a wireless channel-induced packet loss [3][4]. These are sound techniques to improve TCP performance in ad hoc networks, but they rely on global deployment at every node. These techniques may also be difficult to adopt in practice because of the potential heterogeneity of participating nodes. Nodes ranging from high-end notebooks to hand-held devices have varying resources and consequently varying abilities to adopt full-blown network-oriented transport solutions.

In contrast, the end-to-end approach is easy to implement and deploy, requires no network support, and provides the flexibility for backward compatibility. In this paper, we explore an end-to-end approach to improve TCP performance in mobile ad hoc networks. We implement our design only at the two end hosts, and do not rely on any explicit network notification mechanism. End-to-end measurements are used to detect congestion, disconnection, route change, and channel error, and each detection result triggers corresponding control actions.

End-to-end measurement data in ad hoc networks are

[1] Even with link-layer re-transmissions of 802.11 MAC, packet loss still occurs during bursty channel error or MAC-layer contentions.

noisy and consequently may lead to false detections and notifications. Robust detection using noisy measurements poses a great design challenge. Previous study [9] shows that a single metric based approach, e.g., using round-trip time (RTT) or packet inter-arrival time, is not encouraging because of the large amount of noise associated with the measurements. False detection can come in two forms. Network congestion, for example, may go undetected, or conversely congestion may be inferred when the network is in fact *not* congested. Using measurements at the end host, the probability that congestion will go undetected is very low. Measurement metrics such as RTT or inter-arrival time indeed increase when the network is congested. However, using single metric measurements, the probability of false congestion detection in an uncongested ad hoc network is quite high. This sort of false detection can lead to serious throughput degradation. The key innovation proposed in this paper is the use of multi-metric joint identification. By exploiting the degree of independence in measurement noise of individual metrics, the probability of false identification can be significantly reduced by cross-verification.

In this paper, we first describe the necessary network states in an ad hoc network to be identified by TCP, and then examine metrics that can be measured end-to-end. In particular, two metrics are devised to detect congestion, IDD (Inter Delay Difference) and STT (Short Term Throughput). They each exhibit a unique pattern upon congestion; and in non-congestion states, they are influenced by different network conditions in such a way that their respective measurement noise is largely independent. Our multi-metric joint identification approach is then able to reduce false detection by cross-validation. Extensive network simulations and real test-bed measurements show that this technique is viable for achieving reasonably accurate detection and can improve TCP performance significantly.

The remainder of the paper is organized as follows: Section 2 provides an overview of our design rationale. Section 3 describes detection. The design and implementation of ADTCP are presented in Section 4. Simulation and real test-bed measurement results are given in Sections 5 and 6. Section 7 discusses the related work and we concludes the paper in Section 8.

2 Design Rationale

In this paper, we use end-to-end measurements to identify the presence of various network conditions that if left unchecked, degrade throughput. To achieve this, we must first determine the network states that TCP must monitor. These states should be our identification target. Then we must determine what available end-to-end metrics can be used to accurately identify network state. The goal of the identification algorithm is therefore a mapping from metric measurements to the target states. We discuss the two problems in this section and describe the identification algorithm in Section 3.

2.1 States To Be Identified

To decide what network states should be identified, we first assume a situation in which TCP knows why its packets are being lost and consider what TCP should do to improve its performance. First, if the packet loss is due to congestion, TCP should apply the congestion control mechanisms; but if not, TCP might do better not to slow down and exponentially backoff its retansmission timeout. Knowing whether the current network is congested or not is important. As it turns out, proper congestion identification proves to be the biggest improvement to TCP in ad hoc networks. Second, if the packet is lost due to reasons other than congestion, TCP can benefit if it further knows whether the loss is due to channel errors or network disconnection. If the loss is due to channel errors, a simple retransmission is adequate. However, if it is due to disconnection, some special probing mechanisms might be needed for a prompt transmission recovery upon network reconnection.

Previous research has indicated that identifying the following network states is necessary to improve TCP performance over ad hoc networks.

- CONGESTION (CONG): When network congestion occurs, ad hoc transport should adopt the same congestion control actions as conventional TCP [12]. Here, we define congestion as queue build-up and packets being dropped due to buffer overflow at some nodes. The following states are considered when there is no network congestion:
- CHANNEL_ERR (CHERR): When random packet loss occurs, without slowing down, the sender should re-transmit the lost packet [3][4].
- ROUTE_CHANGE (RTCHG): The delivery path between the two end hosts can change from time to time, with disconnections that are too short-term to result in a TCP timeout. Depending on the underlying routing protocol, the receiver may experience a short burst of out-of-order packet delivery or packet losses. In both cases, the sender should estimate the bandwidth along the new route by setting its current sending window to the current slow start threshold, and initiating the congestion avoidance phase [2][11].
- DISCONNECTION (DISC): When the delivery path is disconnected for long enough to cause a TCP retransmission timeout, instead of backing off exponentially, the sender should freeze the current state of the congestion window and retransmission timeout, and perform periodic probing until the connection is re-established. Once re-established, the actions in RTCHG should be followed [1][2].

Metric	Definition
IDD	$A^{i+1} - A^i - (S^{i+1} - S^i)$, where A^i is the arrival time of packet i and S^i is its sending time from the sender
STT	$N_p(T)/T$, where $N_p(T)$ is the # of received packets during interval T,
POR	$N_{po}(T)/N_p(T)$ where $N_{po}(T)$ is # of out-of-order packets during T
PLR	$N_l(T)/N_p(T)$ where $N_l(T)$ is # of lost packets during interval T

Table 1. Four proposed metrics

Sometimes a packet loss may be caused by the combined effects of multiple network conditions. For example, RTCHG may cause multiple flows to go across a hot spot so that CONG losses occur; or bursty CHERR might cause repeated link layer failures, so that RTCHG or DISC conditions eventually take place. However, since our goal is to be TCP friendly, CONG is given the highest priority in identification. The other three states are considered only if the network is not congested.

2.2 Devising End-to-End Metrics

End-to-end measurement is widely used in TCP. The round trip time (RTT) is maintained by the TCP sender to calculate the retransmission timeout. Previous work uses delay related metrics to measure the congestion level of the network. For example, [2] and [9] used inter packet arrival delay, and [10] uses RTT to estimate expected throughput. A challenge in ad hoc networks is that packet delay is no longer only influenced by network queue length, but also is susceptible to other conditions such as random packet loss, routing path oscillations, MAC layer contention, etc. These conditions make such measurement highly noisy. Rather than pursue any single metric that is robust to all dynamics of the network, we devise four end-to-end metrics that tend to be influenced by different conditions so that the noise independence among them can be exploited by multi-metric joint identification.

Inter-packet delay difference *IDD* Metric *IDD* measures the delay difference between consecutive packets (Table 1). It reflects the congestion level along the forwarding delivery path by directly sampling the transient queue size variations among the intermediate nodes. In Figure 1, time at the sender and receiver sides is shown by vertical arrows. Upon each packet arrival, the receiver calculates the IDD value. The first figure shows that the onset of congestion and the queue length build up process is reflected by a series of IDD samples with increasing values.

In addition, unlike the conventional inter-packet arrival delay (IAD), IDD is unaffected by random channel errors and packet sending behaviors. The second figure of Figure 1 shows that random packet losses can influence the IAD measurement but not IDD. In the third figure, assuming the network is not congested, the packet sending behavior also influences the IAD; by subtracting the sending time difference, IDD is not affected.

However, in an ad hoc network, there are still a number of situations in which *IDD* values might give an incorrect estimation of congestion. For example, *IDD* can be influenced by non-congestion conditions like mobility induced out-of-order packet delivery. In section 3.3, we evaluate the accuracy of using IDD to detect network congestion. The accuracy decreases from 85% to 55% as node mobility speed increases.

Short-term throughput *STT* Compared with IDD, STT is also intended for network congestion identification. However, it provides observation over a time interval T[2], and is less sensitive to short term out-of-order packet delivery than IDD. Therefore, STT is more robust to transient route changes, which can be very frequent in a mobile ad hoc network. However, using STT alone to detect netework congestion can be susceptible to measurement noise introduced by bursty channel error, network disconnections or altering TCP source rates. In section 3, we combine STT and IDD to jointly identify network congestion.

Aside from the above two delay related metrics, we also consider the following two metrics for non-congestion state identification.

Packet out-of-order delivery ratio *POR* A packet is counted as being out-of-order if it arrives after a packet that was sent later than it (by the same TCP sender). The receiver records a maximum sending time for all the received packets from the TCP connection, denoted by T_{maxtx}. Every received packet that has a sending time-stamp less than T_{maxtx} is added into POR. POR is intended to indicate a route change event. During the route switching period, multiple delivery paths exist. Packets along the new path may catch up, and those along the old path are then delivered out-of-order.

Packet loss ratio *PLR* At each time interval $[t, t+T]$, we compute this metric as the number of missing packets in the current receiving window. POR can be used to measure the intensity of channel error.

In this section, we described network states that are important in improving TCP performance in an ad hoc network, and metrics that can be measured end-to-end. In the next section, we study how to identify these states using the four metrics discussed above.

[2]The choice of T provides a trade-off between metric accuracy and responsiveness. In our design, we choose T as half of the RTT.

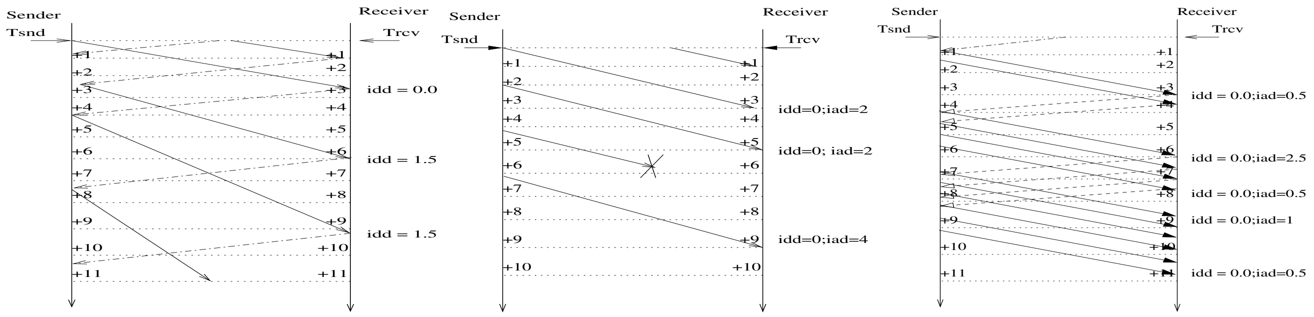

Figure 1. *IDD and IAD Measurements . From left to right: in incipient network congestion, in channel error, in TCP slow start phase*

3 Identification via Multiple Metrics

This section first describes how to detect the four network states based on multi-metric measurements, and then introduces a sample value classification technique to improve the detection accuracy. The achieved accuracy using our approach is also evaluated through simulations.

For all simulation results shown in this section, the default settings are as follows unless otherwise specified. We use the NS-2 simulator with CMU wireless extension modules. 30 wireless nodes roam freely in a $400m \times 800m$ topology following a *random waypoint* mobility pattern, in which the pause time is zero so that each node is constantly moving. The wireless link bandwidth is 2M bps, and IEEE 802.11 and DSR are used as MAC and routing layer protocols. TCP NewReno[12] is used with a maximum window size of eight packets. The packet size is 1460 bytes. Each simulation lasts 300 seconds. To introduce congestion, three competing UDP/CBR flows are run within the time intervals of [50,250],[100,200] and [130,170], respectively. Each UDP flow transmits at 180Kbps.

3.1 Identifying The Network States

3.1.1 Identifying Congestion

To study the relationship between network congestion and IDD/STT, we simulate static and mobile topologies. A TCP flow and three competing UDP/CBR flows are introduced in each simulation to congest the network. The first two figures of Fig. 2 show the simulation results. The first is for the static case without channel errors, and the second is for the mobile case with node mobility speed of 5m/s and 5% channel error. In both figures, we plot the measured IDD/STT values with respect to the instantaneous maximum buffer occupancy of all nodes in the network, which reflects the network congestion level at the sampling time instance. [3]

In Figure 2, we observe that when the maximum network queue size exceeds half of the buffer capacity (25 packets), IDD is clearly *high* and STT clearly *low*. We formalize this observation by defining a value to be HIGH or LOW if respectively it is within the top or bottom 30% of all samples[4]. However, when the network queue size is small (non-congestion case), both IDD and STT vary from LOW to HIGH. Comparing the left two figures of Fig. 2, when node mobility is present, the two metrics become much more noisy in the non-congestion state (i.e., small network queue).

In the single metric-based detection using either IDD or STT, the noise reduces the accuracy significantly when the network is not congested, especially in scenarios with mobility and channel error. In the joint detection approach, we use both metrics to *verify* each other to improve the accuracy. Specifically, we identify a congestion state when both IDD is HIGH and STT is LOW, and non-congestion state if otherwise. The following shows why the multi-metric approach has better detection accuracy than the single metric approach.

When the network is congested, let P_1 and P_2 be the probabilities that IDD is HIGH and STT is LOW respectively. The single metric accuracy is $acc_{idd}(cong) = P_1$ and $acc_{stt}(cong) = P_2$. For the multiple metric case, $acc_{multi}(cong) = P_1 \cdot P_2$. Since the simulations show that $P_1 \simeq P_2 \simeq 1$ (see left two figures of Fig. 2), these three accuracies are roughly equal in congestion state. On the other hand, when the network is not congested, let P_1' and P_2' be the probabilities that IDD is still HIGH and STT is still LOW. Similarly, we have $acc_{idd}(non\_cong) = 1 - P_1'$, $acc_{stt}(non\_cong) = 1 - P_2'$ and $acc_{multi}(non\_cong) = 1 - P_1' \cdot P_2'$. Since each noise probability, $0 < P_1', P_2' < 1$, is non-negligible, multiple metrics thus achieve higher accuracy. Combining these two cases, multi-metric identification improves the accuracy in non-congestion states while

[3]The maximum buffer size for each node is 50 packets in our simulations.

[4]This threshhold was determined empirically from simulation results and real testbed measurements.

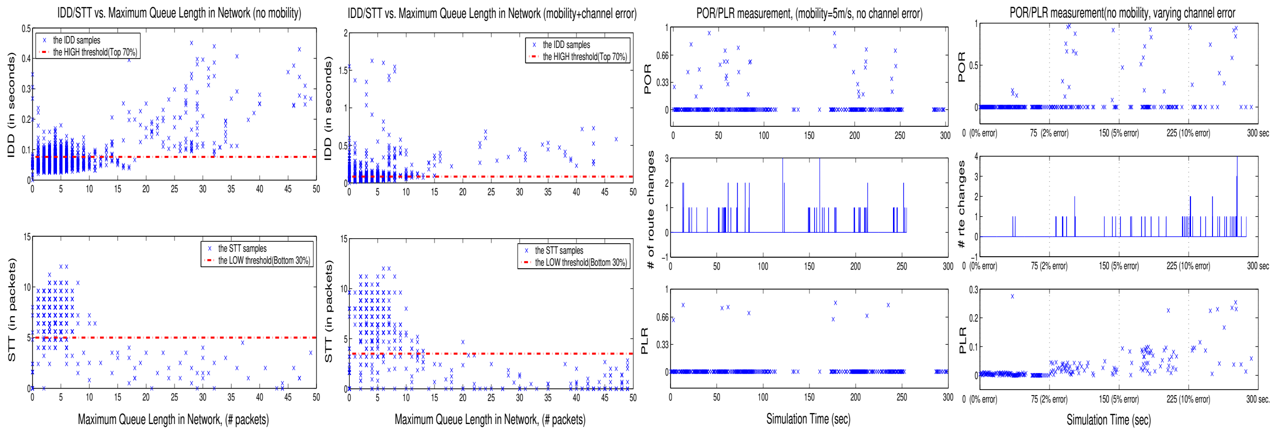

Figure 2. *End To End Metric Measurement. Left two: the IDD and STT measurement w.r.t. instantaneous maximum queue occupancy of all wireless nodes in the network. First figure: static nodes; second figure: mobile nodes (5m/s). Right two: POR and PLR measurements w.r.t. the number of route changes. The third figure: mobile nodes (5m/s), no channel error. The fourth one: static nodes, progressively increasing channel error.*

maintaining a comparable level of accuracy in the congestion state. Therefore, it achieves better identification performance over a variety of network conditions.

The key insight here is that in the non-congestion state, IDD and STT are influenced differently by various network conditions, such as route change and channel error; while in congestion state, they are both dominated by prolonged queueing delay. Thus, the two noise probabilities P_1' and P_2' become largely independent. Effective verification across multiple metrics is possible so long as these conditions do not co-exist during the measurement time interval. Although this joint identification technique cannot achieve perfect accuracy, it does increase accuracy significantly as we show later in Section 3.3

3.1.2 Identifying Non-Congestion States

RTCHG and CHERR If the network state is not congestion, we next seek to detect whether it is RTCHG or CHERR. The third figure of Fig 2 shows data from a simulation run with node mobility speed being 5m/s. A single TCP flow is created without any competing flows. We plot POR and PLR sample values together with the number of route changes in the forwarding path over time. A clear correlation is found between route change events and bursts of high POR measurement. Moreover, since there is no congestion or channel errors in these simulations, PLR remains stable with a few significant outliers. These anomalies correspond to situations in which packets along the old path are excessively delayed or lost.

In the simulation shown in the fourth figure of Fig 2, nodes are immobile and increasing channel error rates (0%, 2%, 5% and 10%) are introduced. In this case, the PLR gradually increases as the channel error rate increases. Note that a high channel error rate can also create route change in the network that will in turn result in bursts of high POR measurements.

	IDD and STT	POR	PLR
CONG	(High, Low)	*	*
RTCHG	NOT (High, Low)	High	*
CHERR	NOT (High, Low)	*	High
DISC	(*, ≈ 0)	*	*
NORMAL	default		

Table 2. *Metrics patterns in 5 network states. High: top 30% values; Low: bottom 30% values; '*': do not care*

In conclusion, a burst of high POR sample values is a good indication of a route change and a high PLR is a good indication of a high rate of channel error. It should be noted that the network may be both in a state of high channel error and route change, which can be identified by high values in both PLR and POR.

DISC Disconnection happens when packet delivery is interrupted for non-congestion reasons for long enough to trigger a retransmission timeout at the sender. If the sender maintains the previous network state estimation, a DISC state can be identified at the sender if the current state estimation is non-congestion when retransmission timeout is triggered.

Table 2 summaries the metrics patterns in five different network states. They are the identification rules used in ADTCP. We show later in this section that such an identification method, combined with a simple sample classification technique, achieves an accuracy above 80% on average in all simulations scenarios.

3.2 Sample Value Classification

Given the above identification rules, the next issue is how to tell whether a sampled value of a specific metric is HIGH or LOW. TCP NewReno uses an exponentially weighted moving average method to smooth RTT. We could also apply the same technique on the sample values of each metric and use a threshold value to judge whether the current sample is HIGH or LOW. However an absolute threshold does not work because network conditions change over time, the judgment that a sample value is HIGH or LOW should be made relative to a recent history. To capture this relative measure, we propose a simple density-based technique (RSD-Relative Sample Density) to infer whether a sample value is HIGH or LOW given a set of history records.

Assume sample values x vary in the range $[0, R]$. RSD divides the range R into N intervals, $I_1, I_2, \ldots, I_N$, where interval I_i holds sample values within $[(i-1)R/N, iR/N)$. Denote the total number of samples as S and the number of samples within interval I_i as $s(I_i)$. Given x, its corresponding interval is $I_x = \frac{x}{\lfloor R/N \rfloor} + 1$. To decide how HIGH x is, RSD calculates the ratio of sample values below x over the total number of samples:

$$RSD(x) = \frac{\sum_{i=1}^{x-1} s(I_i)}{S},$$

which is the CDF distribution at x. Given $RSD(x)$, we can readily tell what percentage of sample values is lower than x. An RSD value close to one infers that x is HIGH w.r.t. the history records.

In order that the RSD value reflect current network conditions, we need to maintain an updated history record. To this end, a *forgetting* mechanism is applied. When sample x is taken, we increment its corresponding counter $s(I_x)$ by one, and calculate its RSD value. Then we subtract d_i proportionally from each interval counter $s(I_i)$ so that $\sum_{i=1}^{N} d_i = 1$ and $d_i = I_i/S$. After *forgetting*, the original sample distribution is maintained while the total number of samples S is kept constant. This way we give more weight to the new samples and exponentially less weight to older ones when calculating RSD. An algorithm for RSD calculation and history maintenance is shown [15]. The algorithm needs $O(N)$ memory space, $O(1)$ computations for RSD calculation and $O(N)$ for history maintenance. A detailed comparison between RSD and Weighted Average estimation is also presented in [15].

3.3 Identification Accuracy

We now apply the RSD classification and study the accuracy of our multi-metric congestion identification. In particular, we compare the single-metric (using only IDD or STT) and multiple-metric (using both) approaches. We run two sets of simulations under non-congested and congested cases (Figure 3). In the first non-congested case, a single TCP flow is created within the topology. In the second congested case, three competing UDP flows are created as before. In both cases, 1% random channel error is introduced and the mobility speed is varied from 0 to 20 m/s. We repeat simulations 50 times at each speed to reduce the impact of random topology factors.

During the simulation, upon each packet loss, we compare the identified network state and the actual network state to determine the accuracy of detection[5]. In particular, if a packet is lost due to network congestion, but the algorithm gives non-congestion estimation, we count it as an incompatible error because this error in detection (and only this one) causes ADTCP to be more aggressive than TCP and consequently TCP-incompatible.

Figure 3 shows the percentage of inaccurate identification in both cases. In the single TCP flow case (the left figure), mobility and channel error are the dominant reasons for packet loss. The increase in mobility speed reduces the accuracy of the single-metric identification quickly. However, from the simulations, the multi-metric approach results in only 5% to 20% inaccurate identification. This is achieved by the cross verification between IDD and STT measurements to eliminate false congestion alarms. Meanwhile, the incompatible error remains less than 2%.

In the multi-flow cases (the right figure), congestion happens more frequently. For multi-metric identification, more than 95% accuracy is observed in all simulations with less than 2% incompatible errors. For the single metric approach, accuracy is only about 70% to 80%.

In summary, we have demonstrated that multiple metrics combined with RSD is a feasible approach to detect network events using end-to-end measurements only.

4 ADTCP Design and Implementation

ADTCP seeks to maintain backward compatibility with TCP NewReno. It uses identical connection establishment and teardown processes. It adopts the same window-based congestion control at the sender when network congestion is identified. To improve the performance of TCP NewReno in ad hoc networks, ADTCP makes several extensions at both sender side and receiver side.

[5]The real network state is obtained by a global monitor implemented in NS-2 simulator. See [6] for implementation details.

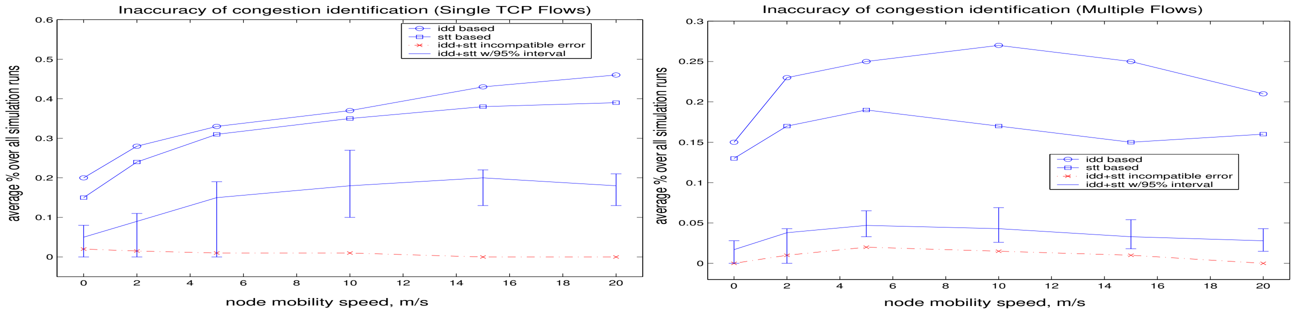

Figure 3. *Identification Accuracy. Left: Percent of inaccurate identifications in a non-congested case, Right: Inaccuracy ratio in a congested case*

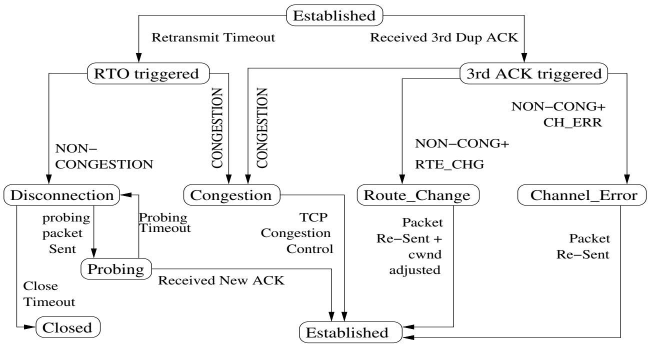

Figure 4. *Modified TCP state machine for ADTCP sender in Linux implementation*

Upon each packet arrival at the receiver, besides the normal operations, values for the four previously discussed metrics are calculated and network states are estimated. The receiver passes state information to the sender in every outgoing ACK packet. This is similar to a soft-state approach. Information about persistent network states will likely be relayed in multiple ACK packets, thus providing robustness against possible ACK lost. The sender maintains the most current state feedback received via these ACK packets. It proceeds with normal TCP operations until an interruption is triggered by either a retransmission timeout or a third duplicate ACK. The sender then takes control actions according to the receiver's network state estimation. A modified TCP state machine is shown in Figure 4 for the sender. In particular, a probing state is introduced to explicitly handle network disconnection. When a non-congestion induced retransmission timeout occurs at the sender, ADTCP freezes its current transmission state and enters a probing state. The sender leaves the probing state when a new ACK is received or the probing is timed out[6]. The TCP connection is closed after multiple failed probing attempts. Due to the space limitation, the detailed pseudo code of ADTCP sender and receiver are omitted here. They can be found in [15].

[6]A similar probing mechanism was proposed by [1]

Discussion It should be noted that the control actions adopted at the sender are not necessarily *optimal*, but are a big step in the direction of improving TCP's operation over ad hoc networks. The sender only takes these control actions upon a third duplicate ACK or retransmission timeout, regardless of the network state feedback received between such events. In our design, receiver-side identification is treated as an enhancement to TCP for ad hoc networks that may help the sender to take more appropriate control and improve transport performance significantly. Without these enhancements, ADTCP would behave exactly as NewReno.

Both the ADTCP sender and receiver are able to work correctly with conventional TCP end hosts. An ADTCP sender communicating with a conventional receiver will receive no explicit network state information in ACK packets and will consequently fall back to standard TCP congestion control operation. On the other hand, a conventional TCP sender that receives the state information bits sent by an ADTCP receiver in an option field of ACK will simply ignore this information. This backward compatibility requires the use of an option field in the TCP header. We discuss this detail in the next section.

4.1 Linux Implementation

We have implemented ADTCP in Linux kernel 2.2.16 with most of the TCP/IPv4 code unchanged. Some implementation details are discussed here.

Receiver Side The identification module is plugged into the receiver side. Kernel space for 4xN integers is allocated for storing metrics samples, where N is set to eight. RSD related calculations are performed after normal processing of each incoming data packet. The network state is represented using 4 bits, with each bit corresponding to one identified state. We introduce an option field in the TCP header

```
+-----------+-----------+------------+
|kind=adtcp |   len=3   |4bits state |
+-----------+-----------+------------+
```

and set the corresponding bits in each outgoing ACK packet.

Sender Side Logic to process the ADTCP option field of a TCP header is introduced at the sender side to read in network state bits from incoming ACK packets. We extend the code that handles third duplicate ACKs and retransmission timeouts to follow the ADTCP design. In particular, when a sender goes into the probing state, it caches its current transmission state and begins using small packets (8 bytes payload) to probe the receiver until it receives an acknowledgement of the reception of the probing packet. Upon leaving the probing state, the previous transmission state is then restored.

Implementation Complexity The complexity of our protocol lies in the metrics collection and state identification process at the receiver side which account for 2/3 of total code added to the Linux kernel. Altogether, ADTCP adds about 300 lines of C code into the IPv4 protocol stack. This code generates additional computational cost on the order of $O(N)$ for each packet reception, and additional memory space of $O(N)$ for each TCP connection. N is the RSD parameter that is set to 8 in practice (refer to section 3.2). This is acceptable for an ordinary user, considering the rapid increase in the CPU power of wireless mobile devices.

5 Performance Evaluation

This section evaluates the performance of ADTCP relative to TCP NewReno and TCP ELFN (as a reference system) through extensive simulation. Instead of using end-to-end measurements, since TCP ELFN collects link state information directly from the network it is expected to be more accurate. A performance gain close to ELFN indicates the effectiveness of ADTCP.

We further examine where ADTCP's performance improvement comes from. ADTCP should not steal a competing TCP flow's fair share of bandwidth and the TCP friendliness should be maintained in all cases. This property is critical to allow ADTCP to be incrementally deployed and benefit the overall network-wide performance.

5.1 Performance Improvement

Figure 5 shows the performance gain of ADTCP compared to NewReno and ELFN using one TCP flow. The simulation parameters are set as is described in Section 3. In all three cases, ADTCP provides significantly better throughput than NewReno and achieves throughput close to ELFN. When nodes are mobile, ADTCP achieves a throughput improvement from 100% to 800% over NewReno.

In Figure 5, it can be seen that NewReno's performance is more sensitive to node mobility than ADTCP. This is because as the mobility speed increases, network disconnection becomes more common. By correctly identifying non-congestion packet losses, ADTCP is able to recover from such interruption quickly and achieve higher throughput.

The presence of channel error slightly increases the performance gap between ADTCP and ELFN (second of Figure 5). The gap comes from the identification inaccuracy of ADTCP. When both mobility and channel error are present, metric samples such as IDD and STT become highly noisy, which makes end-to-end network state detection, especially for non-congestion states, more difficult. In the third figure in which the TCP flow competes with UDP/CBR flows, congestion becomes more frequent and the throughput gap between ADTCP and ELFN reduces. This shows that our identification algorithm detects network congestion state more accurately than non-congestion states.

5.2 TCP Friendliness

ADTCP improves the performance of NewReno because it differentiates packet loss reasons and reacts appropriately. However the TCP friendliness could be a problem if the identification yields too many *incompatible* errors (see Section 3.3) causing ADTCP to react too aggressively.

In order to evaluate ADTCP's TCP-friendliness, we run simulations with four TCP flows (Figure 6). We keep the senders and the receivers of these flows static and place them at the edges of the network topology, so that they all run across some bottleneck area of the network and share the bandwidth with each other[7].

The first figure shows the aggregate throughput of NewReno flows 1,2 and NewReno flows 3,4. The aggregate throughputs of all four flows are also shown in the figure. The third figure is for four ADTCP flows. These two sets of simulations show that ADTCP improves the throughput at the network aggregate level. The bandwidth sharing of identical flows is not perfectly fair in these two cases. This is mainly due to the topology edge effects and MAC-layer's unfairness [8]. In the second figure, we run simulations with flows 1,2 being TCP NewReno and flows 3,4 being ADTCP. Comparing the first and the second set of simulations, we observe that the aggregate throughput of two NewReno flows does not decrease in the two traffic environments. This shows that ADTCP does not steal badwidth from competing TCP flows, and that the performance improvement instead comes from ADTCP's more efficient utilization of the network resources.

[7] According to our simulations, the mobile nodes tend to gather around center of the topology with the random way point mobility pattern. Therefore randomly choosing a sender and receiver often results in short delivery paths (1 or 2 hops), which reduces the chance of bandwidth sharing among competing flows

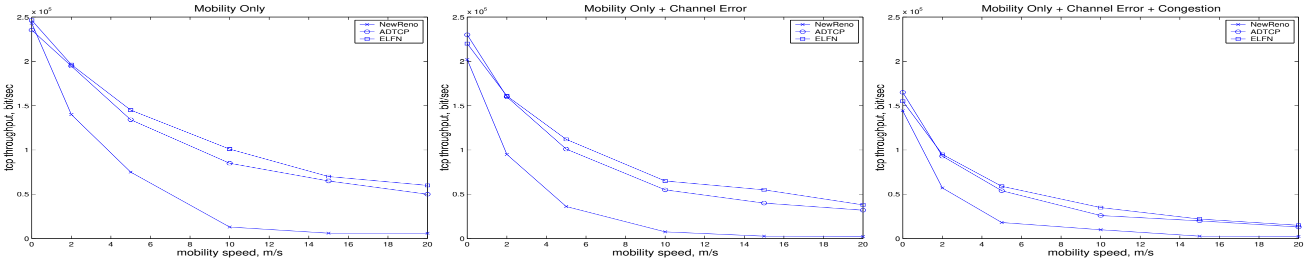

Figure 5. *Performance Improvement of ADTCP. From left to right:1)mobility only, 2)mobility+5% channel error, 3) mobility+5% channel error + 3 competing UDP/CBR flows*

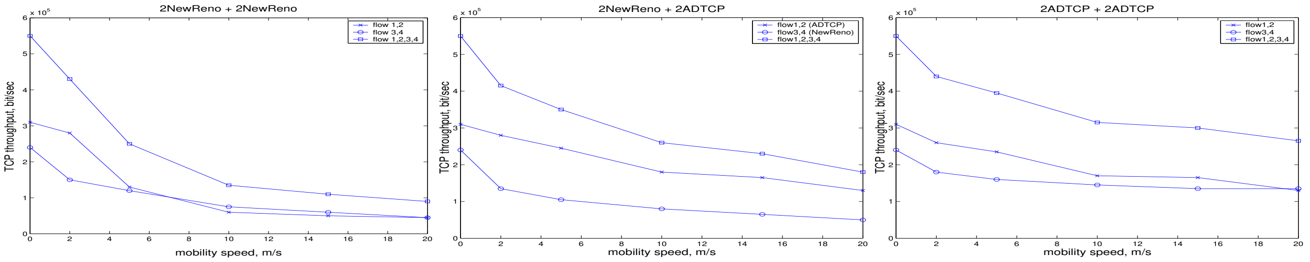

Figure 6. *TCP Friendliness of ADTCP (5m/s Mobility+5% Channel Error). From left to right:1)4 NewReno flows, 2) 2 NewReno + 2 ADTCP flows 3) 4 ADTCP flows,*

6 Real Hardware Measurements

We perform measurements for our ADTCP implementation in our testbed; the testbed configuration is described as the following: Two TCP flows run over a four-hop ad hoc network. The nodes move at a walking speed (around 1-2m/s). Two laptops are P-II portables equipped with WaveLAN wireless cards operating at 2Mb/s. The cards run in ad hoc mode with the RTS/CTS option turned on. Manual routing is configured among the wireless nodes. The TCP parameters are identical to our simulation settings.

We run ADTCP at the receiver side and run both ADTCP and TCP Reno implemented in Linux 2.2.16 at the sending host. This way we are able to test the performance of ADTCP and Reno TCP simultaneously[8]. FTP sessions are created over both TCP connections. The tests run until a 27MB file is transferred by both sessions. We run three sets of test with results shown in Figure 7.

Clear Channel In the first test, the potential performance overhead of ADTCP in a clear-channel, no-mobility, mild congestion with two competing TCP flows scenario is measured. We started the NewReno and ADTCP connections at the sender host almost simultaneously and collected the output traces using *tcpdump*. From the first of Figure 7, we observe that ADTCP performs comparably with Reno TCP. This shows that ADTCP is able to share bandwidth fairly with other conventional TCP connections. A less than 5% decrease in throughput is attributed to its slightly higher computational cost.

Weak Channel Signal We then moved the receiving host farther away from the rest of mobile nodes, so that the signal is weak and channel error increases. We start both TCP Linux and ADTCP simultaneously, and the traces are plotted in second figure. We observe 30% throughput increase of ADTCP over Reno TCP.

Mobility Finally, we test the impact of node mobility. We took a portable receiver and walked around to create network disconnections and reconnections while keeping the other nodes static. From the third figure, we observe more than 100% performance gain of ADTCP over TCP Linux. From the trace, we observe that multiple disconnections happen during the file transfer, ADTCP recovers faster than Reno TCP once the network become reconnected.

7 Related Work

There are two types of approaches in detecting network congestion in the Internet. One is based on end-to-end measurement and the other on feedback from intermediate gateways in the network. Standard TCP [12][10] uses end-to-end measurement of RTT and packet loss to detect

[8]Note that in our tests, we are using a Reno TCP sender and an ADTCP receiver to create a Reno TCP connection since ADTCP is backward compatible.

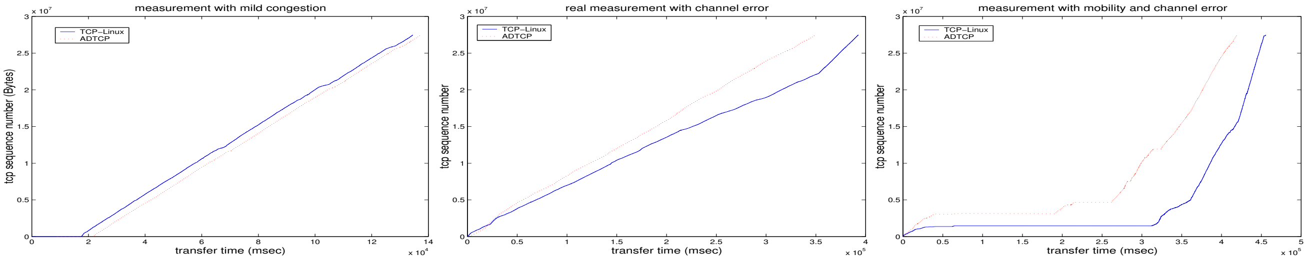

Figure 7. *Real Hardware Measurements. Two TCP Flows, one ADTCP, one Reno TCP in Linux 2.2.16. From left to right:1) Clear channel, 2) Weak signal (high channel error rate), 3) Mobility with varying channel error*

congestion; RED/ECN [5] provides congestion notification by monitoring the instantaneous queue size at the network gateways.

For wireless networks, detecting network congestion is critical in improving TCP performance. In cellular networks, [14][3] directly monitor packet loss at the base station so that wireline packet losses (congestion related) and wireless packet losses (channel error related) are treated differently. As a different approach, [2] [13] propose receiver based measurements on packet interarrival time and loss behaviors to distinguish between congestion losses and wireless link losses.

In mobile ad hoc network, beside wireless channel error, mobility induced link failure occurs more frequently. [1] proposes an explicit link failure notification (ELFN) mechanism for each wireless node to inform the TCP sender. This way the sender can distinguish link failure losses from congestion losses. [7] further shows that ELFN improves standard TCP by as much as seven times in mobile ad hoc network, with about a 5% degradation in static ad hoc networks. For end-to-end measurement based congestion detection, [9] studied the approach of using single metrics measurement such as inter-arrival delay, throughput or packet losses to identify network congestion, but their results are not encouraging. There is simply too much noise in the end-to-end single metric measurement, especially when node mobility and channel errors are both present. In this paper, we further explore the feasibility of end-to-end based congestion detection using *multi-metric joint identification*.

8 Conclusion

The fundamental problem of transport protocol design in mobile ad hoc networks is that such networks exhibit a richer set of behaviors, including congestion, channel error, route change and disconnection, that must be reliably detected and addressed. Detection is challenging because measurement data is noisy. Existing approaches typically rely on network-layer notification support at each router. This paper explores an alternative approach that relies solely on end-to-end mechanisms. To robustly detect network states in the presence of measurement noise, we propose a multiple-metric based joint detection technique. In this technique, a network event is signaled only if all the relevant metrics detect it. Our simulations and real testbed measurements show that ADTCP is able to significantly reduce the probability of false detection while keeping the incompatible detection errors low, and thus greatly improves the transportation performance in a TCP friendly way. This demonstrates that the end-to-end approach is also viable for ad hoc networks.

References

[1] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," *MOBICOM'99*.

[2] P. Sinha, N. Venkitaraman, R. Sivakumar and V. Bharghavan, "WTCP: A reliable transport protocol for wireless wide-area networks," *MOBICOM'99*.

[3] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, "Improving TCP/IP performance over wireless networks," *MOBICOM'95*.

[4] H. Balakrishnan, and R. Katz, "Explicit loss notification and wireless web performances," *Globecom'98*.

[5] S. Floyd, "TCP and explicit congestion notification," *ACM CCR*, 1994.

[6] Zhenghua Fu, Xiaoqiao Meng, Songwu Lu "How bad TCP can perform in wireless ad hoc network" IEEE ISCC (IEEE Symposium on Computers and Communications) 2002, Italy, July 2002.

[7] J. Monks, P. Sinha and V. Bharghavan, "Limitations of TCP-ELFN for ad hoc networks," *MOMUC'00*.

[8] M. Gerla, K. Tang, and R. Bagrodia, "TCP performance in wireless multihop networks," *WMCSA'99*.

[9] S. Biaz and N.H. Vaidya, "Distinguishing congestion losses from wireless transmission losses" *IEEE 7th Int. Conf. on Computer Communications and Networks*, October 1998.

[10] L. Brakmo, S. O'Malley, and L. Peterson "TCP Vegas: New techniques for congestion detection and avoidance" *ACM SIGCOMM 1994*

[11] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", *ACM Mobicom 2001*

[12] M.Allman, V.Paxson, and W. Stevens, "TCP Congestion Control" *RFC 2581* April 1999.

[13] S. Biaz, N. Vaidya, "Discriminating congestion losses from wireless losses using inter-arrival times at the receiver" *IEEE Symp. ASSET'99* March 1999

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

[15] Zhenghua Fu, Ben Greenstein, Songwu Lu, "Design and implementation of a TCP-Friendly transport protocol for ad hoc wireless networks" UCLA Technical Report, 2002. Available at http://www.cs.ucla.edu/wing