# ATP: A Reliable Transport Protocol for Ad-hoc Networks *

Karthikeyan Sundaresan,† Vaidyanathan Anantharaman,§
Hung-Yun Hsieh,† and Raghupathy Sivakumar†

† School of Electrical and Computer Engineering
Georgia Institute of Technology
{sk,hyhsieh,siva}@ece.gatech.edu

§ Wipro Technologies
India
vananth@cisco.com

## ABSTRACT

Existing works have approached the problem of reliable transport in ad-hoc networks by proposing mechanisms to improve TCP's performance over such networks. In this paper we show through detailed arguments and simulations that several of the design elements in TCP are fundamentally inappropriate for the unique characteristics of ad-hoc networks. Given that ad-hoc networks are typically stand-alone, we approach the problem of reliable transport from the perspective that it is justifiable to develop an entirely new transport protocol that is not a variant of TCP. Toward this end, we present a new reliable transport layer protocol for ad-hoc networks called ATP (ad-hoc transport protocol). We show through *ns2* based simulations that ATP outperforms both default TCP and TCP-ELFN.

## Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

## General Terms

Algorithms, Design, Performance

## Keywords

Ad-hoc networks, Transport layer, Reliability, Rate adaptation

## 1. INTRODUCTION

Ad-hoc networks are uniquely characterized by the several factors that differentiate them from traditional computer networks: (i) *Lack of a fixed infrastructure:* Due to absence of dedicated routers, mobile hosts in ad-hoc networks also serve as peer-to-peer relays for connections in the network. (ii) *Mobility:* All hosts in the network are mobile, and hence the network topology can be highly dynamic. From the perspective of a single end-to-end connection, not only are the end-hosts mobile, but the intermediate "routers" are mobile too. (iii) *Shared channel:* Because of the all-wireless nature of ad-hoc networks, not only do flows in the same vicinity contend with each other, but part of a flow traversing multiple hops can contend with other parts of the same flow in its vicinity. (iv) *Limited bandwidth:* While mobile hosts in general can be assumed to possess fewer amounts of resources than their static (wireline) counterparts, the wireless channel bandwidth is also scarce, resulting in multi-hop flows typically enjoying limited bandwidths of at most a few hundred kilobits per second.

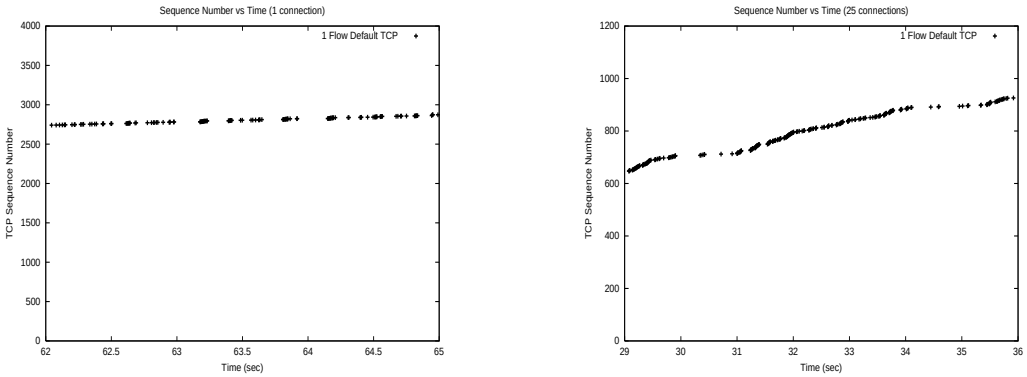
Not surprisingly, such constraining characteristics render traditional wireline network protocols at the different layers of the protocol stack inappropriate for use in ad-hoc networks. At the medium access control (MAC) layer, protocols tailored for wireless environments such as carrier sense multiple access with collision avoidance (CSMA/CA) easily outperform traditional protocols such as carrier sense multiple access (CSMA). At the network layer, numerous routing protocols such as dynamic source routing (DSR) [1], ad-hoc on-demand distance vector (AODV) routing [2], etc., have been proposed for ad-hoc networks. Since such protocols are specifically tailored for the unique characteristics of ad-hoc networks, they significantly outperform conventional wireline routing protocols in an ad-hoc network environment.

At the transport layer, several works have focused on both studying the impact of using transmission control protocol (TCP) as the transport layer protocol, and improving its performance either through lower layer mechanisms that hide the characteristics of ad-hoc networks from TCP, or through appropriate modifications to the mechanisms used by TCP [3–10]. Given the almost universal use of TCP as the transport layer protocol in the current Internet, such works are clearly warranted. However, several applications of ad-hoc networking, including more promising ones such as military battlefields, disaster relief operations, etc., are environments where a completely revamped protocol stack tailored to the operating conditions is not merely feasible, but also justifiable.

In this paper, we approach the problem of providing reliable transport over ad-hoc networks from the above perspective. We argue that TCP or a minor variant of it is not appropriate for the operating conditions common in ad-hoc networks. We study the suitability of the different mechanisms used by TCP for congestion control and reliability, for the characteristics of the target environment. We discuss why a majority of the properties and mechanisms of TCP including window based congestion control, slow-start, loss based congestion detection, multiplicative decrease of congestion

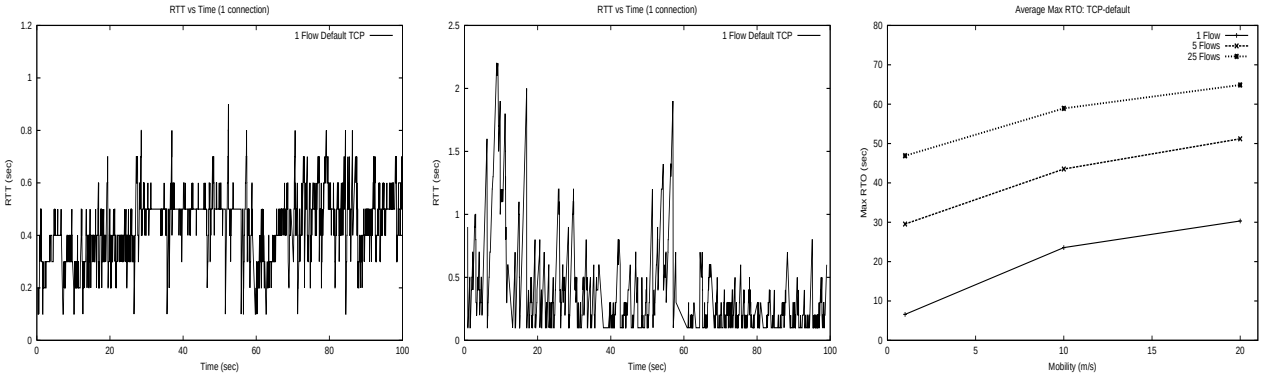
*This work was funded in part by NSF grants ANI-0117840 and ECS-0225497, Yamacraw, and the Georgia Tech Broadband Institute.


*MobiHoc'03*, June 1–3, 2003, Annapolis, Maryland, USA.




(a) [1 Flow, 1m/s] (b) [25 Flows, 1m/s]

**Figure 1: TCP burstiness**




(a) [1 Flow, 1m/s] (b) [25 Flows, 1m/s] (c) RTO values

**Figure 2: TCP RTT samples and RTO values**

window, use of retransmission timeouts, and reliance on reverse path characteristics, are not fundamentally appropriate for an ad-hoc network.

We then present a new transport protocol called ATP (ad-hoc transport protocol) that is tailored toward the characteristics of ad-hoc networks. ATP, by design, is an antithesis of TCP and consists of: rate based transmissions, quick-start during connection initiation and route switching, network supported congestion detection and control, no retransmission timeouts, decoupled congestion control and reliability, and coarse grained receiver feedback. We present the details of the ATP algorithm and show through simulation results that it significantly outperforms both TCP and TCP-ELFN (TCP variant with explicit link failure notification) [6] under a variety of load and mobility conditions.

The rest of the paper is organized as follows: In Section 2, we discuss the shortcomings of the TCP design in the context of the ad-hoc network environment. In Section 3, we outline the design goals and key components of ATP, while in Section 4, we present the details of the ATP protocol. In Section 5, we compare ATP with TCP and TCP-ELFN. In Section 6 we discuss some related work, and finally in Section 7, we conclude the paper.

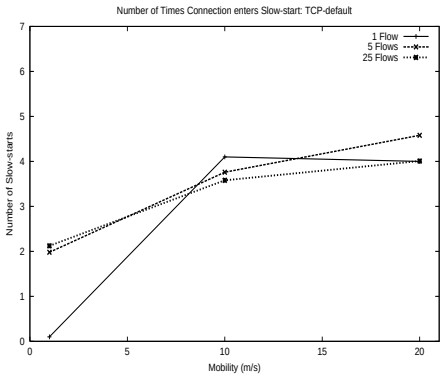
# 2. MOTIVATION

In this section, we discuss in detail the appropriateness of the different design elements in TCP in the context of an ad-hoc network environment. Specifically, we categorize the discussion based on the following components and characteristics of TCP: (i) Window based transmissions, (ii) Slow-start, (iii) Loss based congestion detection, (iv) Multiplicative decrease, (v) Use of retransmission timeouts, and (vi) Reliance on reverse path. When necessary, we use simulation results to substantiate and illustrate our arguments. While we present details of the simulation model in Section 5, briefly: the *ns2* network simulator is used for all the simulations; the network consists a 1000mx1000m network grid with 100 nodes; nodes move randomly using the way-point mobility model with a maximum speed of 1m/s, 10m/s or 20m/s depending upon the scenario; different traffic loads of 1 flow, 5 flows, and 25 flows are used; the application generating the traffic is FTP; source-destination pairs are randomly chosen from the 100 nodes; dynamic source routing (DSR) and IEEE 802.11b in the distributed coordination function mode (CSMA/CA) are used as the routing and MAC protocols respectively[1] and unless explicitly specified, results are averaged over simulations run with 10 different random seeds, and simulations are run for a duration of 100 seconds. We use TCP-NewReno for all our simulations and discussions.

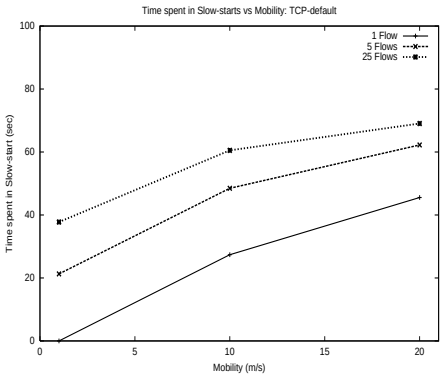
## 2.1 Window Based Transmissions

TCP is a window based protocol. One of the underlying motivations behind such a design choice is to avoid the maintenance of any fine-grained timers on a per-flow basis. For wireline environments, where per-flow bandwidths can scale up to several megabits

[1]While our arguments are agnostic to the specific underlying protocols used, we identify dependencies in the results observed, if any.




(a) Number of slow-starts




(b) Average time in slow-start

**Figure 3: TCP slow-start**

per second, such a design choice is clearly essential. However, the use of a window based transmission mechanism in ad-hoc network networks results in the critical problem of burstiness in packet transmissions.

TCP relies on self-clocking (ACKs arriving to trigger further transmissions) in the absence of timers. Thus, if several ACKs arrive back-to-back at the sender, a burst of data packets will be transmitted by the sender even if it were in the congestion avoidance phase (where one packet will be transmitted for every incoming ACK). Unfortunately, *ACK bunching* or several ACKs arriving at the same time is a norm in ad-hoc networks because of the short-term unfairness of the CSMA/CA MAC protocol typically used in such networks. [11] provides a good exposition on the short term unfairness properties of CSMA/CA. Such short-term unfairness results in the data stream of a TCP connection assuming control of the channel for a short period, followed by the ACK stream assuming control of the channel for a short period. Interestingly, such a phenomenon will occur even when the ACK stream does not traverse the exact same path as the data stream. This is because even if the paths were completely disjoint, the vicinity (2-hop region in the case of CSMA/CA) of the TCP sender and the vicinity of the TCP receiver still are common contention areas for the data and ACK streams.

Figures 1(a) and (b) show the TCP sequence number progression (at the sender) for one TCP connection (say $f_1$ and $f_{25}$) in two scenarios with 1 flow and 25 flows respectively. It can be seen that the transmissions occur in periods of bursts and are interspersed with periods of inactivity due to the arrival of ACKs. The impact of such burstiness of traffic has two undesirable effects:

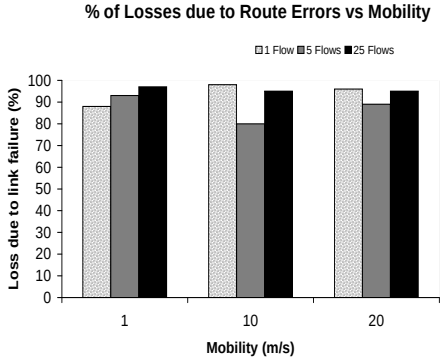
- *Varying round-trip time estimates:* TCP relies on an accurate round-trip time (*rtt*) calculation to appropriately set the timer for its retransmission timeout (RTO). Coupled with the low bandwidths available to flows, the burstiness results in artificially inflating the round-trip time estimates for packets later in a burst. For example, the $i^{th}$ packet in a burst experiences an *rtt* of $rtt_{base} + (i-1) * L/r$, where $rtt_{base}$ is the base *rtt* of the underlying path, $L$ is the length of a packet, and $r$ is the available rate. Essentially, the round-trip time of a packet is impacted by the transmission delay of the previous packets in the burst due to the typically small available rates. Figures 2(a) and (b) present the *rtt* samples observed by $f_1$ and $f_{25}$ over the entire simulation duration. It can be observed that the *rtt* values fluctuate periodically. TCP sets its RTO value to $rtt_{avg} + 4 * rtt_{dev}$, where $rtt_{avg}$ is the exponentially average of *rtt* samples observed, and $rtt_{dev}$ is the standard deviation of the *rtt* samples. Hence, when *rtt* samples vary widely due to the burstiness, the RTO values are highly inflated, potentially resulting in significantly delayed loss recovery (and hence under-utilization). Figure 2(c) presents the average maximum RTO values for the connections in different scenarios.

- *Higher induced load:* Spatial re-use in an ad-hoc network is the capability of the network to support multiple spatially disjoint transmissions. Unfortunately, due to the burstiness and the short term capture of channel by either the data stream or the ACK stream, the load on the underlying channel can be higher than the average offered load. For example, if a flow's instantaneous rate is 10 packets per second, while the ideal inter-packet separation that would allow for optimal use of the underlying channel is 100ms, bursty transmissions can result in higher contention at the MAC layer (recall that downstream hops of the same flow contend with the upstream hops due to the shared channel characteristics). We refer to the artificially (short-term) increased load on the underlying channel as the induced load. If the offered load is not high, the higher induced load will not result in any major performance degradation. However, if the offered load itself is high (around the peak scalability of the underlying MAC layer's utilization curve), the utilization at the MAC layer can suffer significantly.

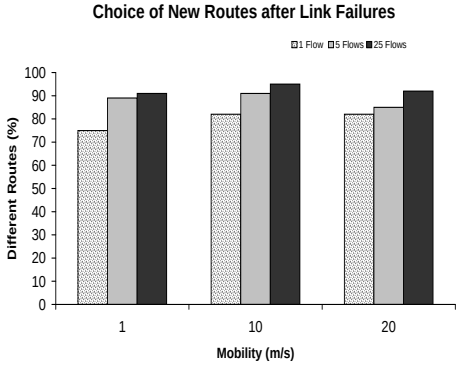
## 2.2 Slow-start

The slow-start mechanism is used by TCP both during connection initiation and when TCP recovers from what it perceives as heavy congestion in the network. For both cases, the goal of slow-start is to *probe* for the available bandwidth for the connection. When a connection is in the slow-start phase, TCP responds with two data packet transmissions for every incoming ACK. While this exacerbates the burstiness problem discussed earlier, there are two other problems associated with the slow-start mechanism in the context of ad-hoc networks:

- *Under-utilization of network resources:* Although slow-start uses an exponential increase of the congestion window size, the increase mechanism is still non-aggressive by design as it can take several *rtt* periods before a connection operates at its true available bandwidth. This is not a serious problem in wireline networks as connections are expected to spend most of their lifetimes in the congestion avoidance phase.




(a) % of losses due to route-failures




(b) % of new routes computed

**Figure 4: Loss based congestion indication**

However, because of the dynamic nature of ad-hoc networks, connections are prone to frequent losses which in turn result in frequent timeouts and hence more slow-start phases. Figure 3(a) presents the average number of times connections enter slow-start during a 100 second simulation for different rates of mobility and different loads. Figure 3(b) presents the average time spent in slow-start by the connections during the 100 second simulation. It can be observed that connections spend a considerable amount of time in the slow-start phase, with the proportion of time going above 50% for the higher loads. Essentially, this means that connections spend a significant portion of their lifetime probing for the available bandwidth in lieu of operating at the available bandwidth.

- *Unfairness:* TCP's fairness properties are firmly dependent upon the contending connections operating in congestion avoidance. When connections operate primarily in the slow-start phase, the fairness properties of TCP are more likely to be violated. In Section 5, we show how connections experience unfairness when using TCP in ad-hoc networks.

## 2.3 Loss Based Congestion Indication

TCP uses the occurrence of losses (inferred either through receipt of three duplicate ACKs, or occurrence of a timeout) to detect congestion. While congestion is by far the main source of losses in wireline networks, it is well known that this is not the case in wireless networks. In conventional cellular wireless networks, non-negligible random wireless channel error rates also contribute to losses, and considerable amount of research has been done toward either hiding such losses from TCP through link-layer reliability [12], or enhancing TCP with mechanisms that can distinguish congestion losses from random errors [13]. In ad-hoc networks, in addition to congestion and random wireless errors, mobility serves as another primary contributor to losses perceived by connections. Random wireless errors are addressed to some extent through the use of a semi-reliable MAC layer such as CSMA/CA that uses a positive ACK after data reception to indicate successful reception of a packet. Interestingly, CSMA/CA does not distinguish between whether a link is down because of the other end moving out of range, or because of high contention at the receiver. In either case, after attempting to transmit to a receiver for a finite number of times, the MAC layer concludes a link failure and informs the higher layers accordingly. Most routing protocols designed for ad-hoc networks [1, 2] rely on such MAC feedback to trigger route-failure notification to the source.

Losses in ad-hoc networks can be classified into either link failure induced, or congestion induced (interface queue overflows). Figure 4(a) presents the percentage of the number of losses due to route (link) failures for different rates of mobility and loads. It can be observed that in all the scenarios, more than 80% of the losses in the network are due to link failures. Note that a link failure can be inferred by the MAC layer even when it is not able to reach a neighbor due to severe congestion. However, irrespective of the true cause of link failure inference, the source will be notified of a route failure and a new route computation will be performed. Figure 4(b) shows the percentage of time when the old route is again chosen by the route computation mechanism. It can be observed that about 90% of the time, a different route is chosen.

Essentially, most losses in ad-hoc networks occur as a result of route failures (in reality, the MAC and routing layer *perceive* most of the losses as due to route failures), and hence treating losses as an indication of congestion turns out to be inappropriate. We elaborate on this factor further in our discussion of the LIMD mechanism used by TCP.

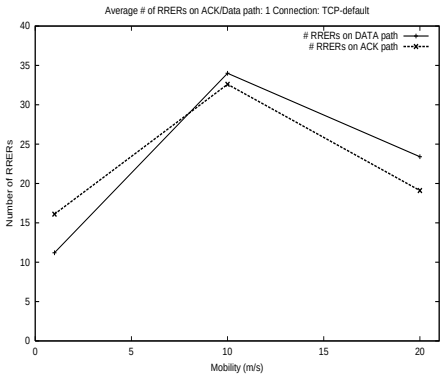
## 2.4 Linear Increase Multiplicative Decrease

The linear increase phase of TCP has the same drawback of slow-start – slow convergence to the optimal operating bandwidth, and hence vulnerability to route failures before the optimal bandwidth is attained. The multiplicative decrease on the other hand is inappropriate for the reasons discussed in Section 2.3. Essentially, most loss events in an ad-hoc network are due to route failures, or are perceived to be due to route failures by the underlying layers. Hence, more often than not, a loss event experienced by a connection is followed up by a route change (see Figure 4(b)). While TCP's multiplicative decrease is an appropriate reaction to congestion, it is definitely not an appropriate action to take when a route change has occurred. Ideally, when a route change occurs, TCP should enter its bandwidth estimation phase as its old congestion window state is not relevant to the new route.

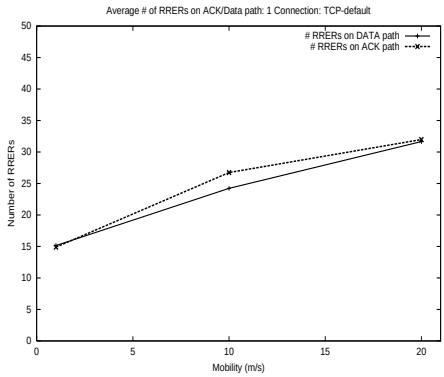
When TCP-ELFN is used, the state of the TCP sender is frozen till a new route is computed. While this prevents undesirable timeouts from occurring, the ELFN mechanism still suffers from the drawback of an inappropriate use of the old congestion window state.

## 2.5 Dependence on ACKs

TCP relies on the periodic arrival of ACKs both to ensure reliability and to perform effective congestion control. Most implementations of the TCP receiver send one ACK for every two packets




(a) 1 Flow




(b) 25 Flows

**Figure 5: Number of route errors**

received. This dependence on ACKs results in two problems for ad-hoc networks: (i) Due to the overhead (about 100 bytes) associated with the request-to-send (RTS), clear-to-send (CTS), and ACK packets used by the CSMA/CA protocol, TCP ACKs sent from the receiver to the sender can amount to 10-20% of the data stream rate. If the forward and reverse paths happen to be the same,[2] the ACK traffic in the reverse path will contend with the data stream on the forward path and reduce the rate enjoyed by the data stream. (ii) If the forward and reverse paths are not the same, the progress of the TCP connection will be dependent on both the forward path and reverse path reliability. Thus, the chances of a connection stalling increase when different paths are used. Note that even if the forward and reverse paths are different, due to the shared channel in the vicinity of the sender and the vicinity of the receiver, the data and ACK streams will still contend with each other. Figures 5(a) and (b) show the number of times the data stream and the ACK stream experience independent path failures for the 1 flow scenario and the 25 flows scenario respectively. It can be observed that the forward and reverse paths experience the same order of magnitude of failures.

# 3. THE ATP DESIGN

In this section, we outline the key design elements of the proposed *ad-hoc transport protocol* (ATP). For each element, we relate the element to problems identified in Section 2.

## 3.1 Layer Coordination

One of the emerging trends in adapting protocols for wireless networks in general and ad-hoc networks in particular is a more involved coordination between different layers of the protocol stack. For example, most routing protocols designed for ad-hoc networks rely on MAC layer information to detect link (and hence path) failures. Mechanisms to further improve performance of routing protocols with additional MAC layer information have been proposed in related work [14]. A further degree of coordination possible in ad-hoc networks is the explicit coordination between the different nodes in the network to improve end-to-end performance. For example, TCP-ELFN uses *link failure notification* from the intermediate routers to freeze TCP's state at the sender.

One of the key cornerstones of the ATP design is the use of lower layer information and explicit feedback from other network nodes to assist in the transport layer mechanisms. Specifically, ATP uses feedback from the network nodes for three different purposes: (i) initial rate feedback for start-up rate estimation, (ii) progressive rate feedback for congestion detection, congestion avoidance, and congestion control, and (iii) path failure notification. While any node coordination can potentially constrain the scalability of a protocol, ATP does not require any per-flow state maintenance at the intermediate nodes, and hence is highly scalable.

[2]Routing protocols in ad-hoc networks may or may not choose the same path in two directions.

## 3.2 Rate Based Transmissions

ATP uses rate based transmissions in lieu of the window based transmissions performed by TCP. Rate based transmissions aid in improving performance in two ways: (i) They avoid the drawbacks due to burstiness identified in Section 2, and (ii) Since the transmissions are scheduled by a timer at the sender, the need for self-clocking through the arrival of ACKs is eliminated. The latter benefit is used by ATP to decouple the congestion control mechanism from the reliability mechanism, and also to alleviate the impact of reverse path characteristics on the performance experienced by the data stream on the forward path.

While an obvious limitation of rate based schemes is the timer overheads incurred at the sender, the timer granularity required for the limited bandwidths in an ad-hoc network is large enough to be realized without significant overheads. For example, with a reasonable load in the network, say 10 flows (or 25 flows), a packet size of 512 bytes, and a raw channel capacity of 2Mbps, the timer granularity required is 40ms (or 125ms).

## 3.3 Decoupling of Congestion Control and Reliability

Unlike in TCP where the congestion control and reliability mechanisms are tightly coupled through dependence on ACK arrival, in ATP the two mechanisms are decoupled. Congestion control, as introduced earlier, is performed using feedback from the network, while reliability is ensured through coarse grained receiver feedback and selective ACKs. While we elaborate on the details of the mechanisms in Section 4, briefly:

- To facilitate congestion control, the intermediate nodes in the network provide congestion information in terms of the available rate. The feedback is piggybacked on the data packets in the forward path, and the ATP receiver consolidates such information and sends back the collated feedback information.

- For reliability, the receiver also uses selective ACKs to report back to the sender any new holes observed in the data stream. Unlike in TCP where the SACK information is complimentary to the cumulative ACK scheme, ATP relies solely on the SACK information.

## 3.4 Assisted Congestion Control

ATP's congestion control protocol relies on feedback from the intermediate nodes traversed by the connection to adapt the sending rate. Briefly, each node in the network maintains two parameters: $Q_t$ (an exponential average of the queuing delay experienced by packets traversing that node), and $T_t$ (an exponential average of the transmission delay experienced by the head-of-line packet at that node). $T_t$ is influenced by the contention experienced between packets within nodes in the same contention vicinity, while $Q_t$ is influenced by the contention between packets belonging to different flows at the same node. For every packet that passes through a node, the node stamps the sum $Q_t + T_t$ if the already stamped sum on the packet is smaller than its current value. The receiver of an ATP connection further performs an exponential averaging of the values stamped on the incoming packets. For every epoch period, the receiver sends rate feedback to the sender using the exponentially averaged value. The sender, based on its current rate, and the rate specified in the feedback determines whether to *increase*, *decrease*, or *maintain* its rate. The maintain phase in ATP is a critical difference from the states that a TCP connection can be in. In addition, the increase and decrease operations performed by ATP are more accurate because of the network feedback received.

## 3.5 TCP Friendliness and Fairness

TCP friendliness is not a constraint under which ATP is designed, since it is targeted for ad-hoc network environments where network nodes will possess a dedicated protocol stack. However, fairness among ATP flows is still of key concern just as in TCP. Since ATP relies on the intermediate network nodes for feedback on congestion, we show in Section 5 that ATP exhibits good fairness properties.

# 4. THE ATP PROTOCOL

In this section, we elaborate on the specific mechanisms used by ATP. Briefly, just as in TCP, ATP primarily consists of mechanisms at the sender to achieve effective congestion control and reliability. However, unlike in TCP, ATP relies on feedback not just from the receiver, but also from the intermediate nodes in the connection path. In terms of specific functionality, the intermediate nodes provide congestion feedback to the sender, while the receiver provides feedback for both flow control and reliability. The receiver also acts as a collator of the congestion information provided by the intermediate nodes in the network before the information is sent back to the sender. The receiver provides the reliability, flow control, and collated congestion control information through periodic messages. The sender on the other hand, is responsible for connection management, start-up rate estimation (with network feedback), congestion control, and reliability.

In the rest of the section, we start with describing the role of the intermediate nodes. We then describe the mechanisms at the ATP receiver for providing rate feedback and sending SACK information to the sender. Finally, we describe in detail the different components of the ATP sender for start-up behavior, congestion control, and reliability.

## 4.1 Intermediate Node

ATP relies on the intermediate nodes that a connection traverses to provide rate feedback information. Intermediate nodes in the network maintain sum of the average queuing delay ($Q_t$) and the average transmission delay ($T_t$) experienced by packets traversing through them. While $T_t$ at a node is impacted by the contention between the different nodes in the vicinity of that node, $Q_t$ is impacted by the contention at that node between different flows traversing the node.

Note that the values $Q_t$ and $T_t$ are computed over all the packets traversing the node, irrespective of the specific flow the packets belong to. Thus, $Q_t$ and $T_t$ are maintained on a per-node basis, and not on a per-flow basis. For every outgoing packet, an intermediate node updates its $Q_t$ and $T_t$ values. The values are maintained using exponential averaging as follows:

$$Q_t = \alpha * Q_t + (1 - \alpha) * Q_{sample}$$

$$T_t = \alpha * T_t + (1 - \alpha) * T_{sample}$$

where $Q_{sample}$ and $T_{sample}$ are the queuing delay and transmission delay experienced by the outgoing packet. We use a value of 0.75 for $\alpha$ in our simulation results. In addition, each packet consists of a rate feedback field $D$ (note that the rate will actually be an inverse of D) that consists of the maximum $Q_t + T_t$ value at the upstream nodes the packet has traversed through. When the packet is dequeued for transmission, the intermediate node checks to see if $D$ is smaller than the $Q_t + T_t$ value at that node. If $D$ is smaller, the intermediate node updates the $D$ on the packet to its $Q_t + T_t$ value.

When the receiver receives a packet, the $D$ field in the packet indicates the maximum (average) delay experienced by packets at any of the intermediate nodes it traversed through.

When a connection starts-up or a *probe* packet is sent by the sender, the intermediate node behavior is the same except when there is no other traffic around the node. When the node observes an idle channel, it uses $\eta * (Q_t + T_t)$ as the delay instead of the normal $Q_t + T_t$. The reasoning for this behavior is as follows. When the channel around the intermediate node is idle, the $Q_t + T_t$ values will be determined by the actual queuing delay and transmission delay experienced by the probe packet. However, when the actual data flow begins, packets belonging to the flow, at every hop in the path, will contend with other packets belonging to the same flow at both upstream and downstream nodes. Intermediate nodes project this rate by appropriately setting $\eta$. For CSMA/CA the typical value for $\eta$ is 3. (In a linear chain, every third node can transmit, but for a large number of hops it can be as high as 5 for a path of length 5.)

Note that ATP's requirement for node-coordination thus constitutes only the maintenance of two parameters $Q_t$ and $T_t$, and appropriately stamping packets being forwarded. Also, the only additional field in the ATP header for the forward (data) path, besides the other fields in the TCP header, is the rate feedback field D.

## 4.2 ATP Receiver

The ATP receiver provides periodic feedback to the sender to assist in its reliability and flow control mechanisms. In addition, it also collates the rate feedback information provided by the intermediate nodes (through the $D$ field on the packets), and sends it back to the sender. In order to send the feedback periodically, the receiver runs an epoch timer of period $E$. Note that the period $E$ should be larger than the round-trip time of a connection, but at the same time must be small enough to track the dynamics of the path characteristics. $E$ is empirically chosen to be one second in our simulations.

### 4.2.1 Rate Feedback

For every incoming packet belonging to a flow, the receiver performs an exponential averaging of the $D$ value specified in the packet:

$$D_{avg} = \beta * D_{avg} + (1 - \beta) * D$$

After every epoch timer expiry, the receiver provides the $D_{avg}$ value at that time as feedback to the sender. The exception is when the flow control rate determined by the receiver (see Section 4.2.3) is smaller than the rate projected from the $D_{avg}$ value. We use a value of 0.75 for $\beta$ in all our simulations.

### 4.2.2 Reliability Feedback

The ATP receiver uses selective ACKs (SACKs) for providing information about losses in the data stream received. Since the feedback is not provided for every incoming data packet, but rather on a periodic basis, ATP uses a larger number of SACK blocks than TCP-SACK. While TCP-SACK uses only 3 blocks per ACK, ATP uses 20 SACK blocks in its reliability feedback. For the typical channel data rate of 2Mbps, the average per-flow rate is about 100Kbps even when there are only 5 connections in the network (see Section 5). For an epoch period of 1 second and a packet size of 512 bytes, the rate translates into about 25 packets per second. Hence, using 20 SACK blocks, most if not all of the losses in an epoch can be identified. However, unlike in TCP where the SACK blocks on ACKs progressively identify newer holes, in ATP the SACK blocks always identify the first sequence of 20 holes in the data stream. This is because ATP does not use a retransmission timeout at the sender and hence has to rely on the feedback from the receiver to perform correct error recovery.

In terms of the reverse path overheads, for the above example, TCP-SACK can be shown to incur an overhead of 2.4Kbps for the SACK blocks (one ACK for every two data packets, three blocks per ACK, two sequence numbers per block, and 4 bytes per sequence number), while ATP would incur an overhead of 1.28Kbps for an epoch period of one second. However, the overhead of ATP can be shown to be considerably smaller if the overall header costs at the transport layer, and at the lower layers are accounted for. This is because ATP aggregates its feedback into one packet, while TCP-SACK will send feedback for every two data packets. A similar notion of feedback optimization is also used in [15], albeit in a different context.

We address how suffix losses (that SACK information cannot identify) are handled at the sender in Section 4.3.3.

### 4.2.3 Flow Control Feedback

Unlike in TCP, where the flow control is achieved through appropriate window advertisements, ATP performs flow control by observing the rate $R_{app}$ at which the application is processing in-sequence data from the receive buffer. When rate feedback is being sent to the sender, if the application read rate ($R_{app}$) is smaller than the rate feedback, the rate feedback information is replaced with the $R_{app}$.

The receiver in its periodic feedback message to the sender (once every epoch), sends both the rate and reliability feedback. We elaborate on the sender side mechanisms that use the feedback provided by the receiver in the next section.

## 4.3 ATP Sender

The ATP sender, like in TCP, consists of most of the driving mechanisms of the transport layer protocol. Specifically, the ATP sender consists of the components for the following functionality: (i) quick-start, (ii) congestion control, (iii) reliability, and (iv) connection management. In the rest of the section, we elaborate on the different components.

### 4.3.1 Quick-start

**Initial Rate Estimation:**

*Sender*

1 send probe packet

*Intermediate node*

2 Compute $Q_t + T_t$ for packet
3 if($Avg(Q_t) + Avg(T_t) > \epsilon$)
4 $Avg(Q_t) = \alpha * Avg(Q_t) + (1 - \alpha) * Current(Q_t)$
5 $Avg(T_t) = \alpha * Avg(T_t) + (1 - \alpha) * Current(T_t)$
6 if ($Avg(Q_t) + Avg(T_t) > stamped\ D$)
7 $stamped\ D = Avg(Q_t) + Avg(T_t)$
8 else
9 $D_{projected} = i * (Current(Q_t) + Current(T_t))$
10 if($D_{projected} > stamped\ D$)
11 $stamped\ D = D_{projected}$

*Receiver*

12 Set $Avg(D) = Current(D)$
13 send $packet_{feedback}$ to sender with $Avg(D)$

*Sender*

14 $packet_{feedback}$ received with $Avg(D)$
15 compute rate $R = \frac{1}{Avg(D)}$
16 $send\_rate\ S = R$
17 $start\_epoch\_timer()$
18 $send\_packet()$

**Figure 6: Pseudo-code for quick-start**

During connection initiation, or when recovering from a timeout, TCP's slow-start mechanism will take a few round-trip times before it can converge on the available bandwidth for a flow. As we show in Section 2, due to the frequent path failures and resultant timeouts in an ad-hoc network, a TCP connection can end up spending a considerable portion of its lifetime in the slow-start phase, thus degrading network utilization.

ATP uses a mechanism called *quick-start* to probe for the available network bandwidth within a single round-trip time. Figure 6 presents the pseudo-code for the quick-start mechanism. Essentially, during connection initiation, ATP uses a TCP-like SYN - SYN+ACK exchange between the sender and the receiver. The intermediate nodes, when they forward the SYN packet, stamp on the packet the $Q_t + T_t$ delay in the manner described in Section 4.1 (lines 2-11 in Figure 6). When the receiver responds back with an ACK, it piggybacks onto the ACK the $Q_t + T_t$ value that was stamped on the incoming SYN packet (lines 12-13). The sender, upon receiving the ACK, starts using the rate value obtained based on the feedback (lines 14-18).

ATP performs the quick-start operation both during connection initiation and when the underlying network path traversed by the connection changes. The motivation for performing quick-start when a path change occurs is straightforward. When a new path is used, the connection is not aware of the available bandwidth on the path. Hence, performing bandwidth estimation once again allows the connection to operate at the true available bandwidth instead of either over-utilizing or under-utilizing the resources available along the new path.

### 4.3.2 Congestion Control

**Normal Operation:**

*Intermediate node*

1 Compute $Q_t + T_t$ for packet
2 if$(Avg(Q_t) + Avg(T_t) > \epsilon)$
3 $Avg(Q_t) = \alpha * Avg(Q_t) + (1-\alpha) * Current(Q_t)$
4 $Avg(T_t) = \alpha * Avg(T_t) + (1-\alpha) * Current(T_t)$
5 if $(Avg(Q_t) + Avg(T_t) > stamped\ D)$
6 $stamped\ D = Avg(Q_t) + Avg(T_t)$

*Receiver*

7 $Avg(D) = \beta * Avg(D) + (1-\beta) * Current(D)$
$On\ epoch\_timer\ expiry$
8 $stamp\ Avg(D) on\ packet_{feedback}$
9 send $packet_{feedback}$ to sender

*Sender*

10 $packet_{feedback}$ received with $Avg(D)$
11 Compute new rate $R = \frac{1}{Avg(D)}$
*Rate Adjustment:*
12 $if\ sendrate S < R - \phi * S$
13 $S = S + \frac{R-S}{k}$
14 $else\ if\ S > R$
15 $S = R$
16 $elsemaintain S$
17 $start\_epoch\_timer()$
18 $send\_packet()$

**Figure 7: Pseudo-code for normal operation**

Unlike TCP, which has a two-phase congestion control protocol with an increase-phase and a decrease-phase, ATP uses a three-phase congestion control protocol consisting of increase, decrease, and maintain phases. One of the key differences between TCP's congestion control mechanisms and that of ATP's is the network feedback that ATP's mechanisms use. Since TCP does not rely on any network support, it probes for more bandwidth by linearly increasing the congestion window size at the sender. Similarly, when a loss occurs, since TCP does not know the true extent of congestion, it conservatively performs a multiplicative decrease of the congestion window size.

ATP, on the other hand, relies on feedback from the intermediate network nodes. Hence, its increase can be more aggressive than that of TCP, decrease can be less conservative than that of TCP, and more importantly can operate in a *maintain* phase when network conditions do not change. We now elaborate on each of the phases in more detail:

- *Increase phase:* When the feedback rate from the receiver is greater than the current rate $S$ by a threshold $\phi * S$, the sender enters the increase phase (lines 12-13 in Figure 7), where $\phi$ is a small constant used to prevent fluctuations. The threshold is kept as a function of the current rate in order to allow contending flows with lower rate to increase more aggressively than the flows with larger rates. Once an increase decision is taken, flows increase their rates only by a fraction $k$ of the potential increase amount. We choose a value of 5 for $k$ in the simulations due to following rationale: When the rate of a flow is increased by one packet per second, the induced load (when the underlying MAC scheme is CSMA/CA) in the network can increase by up to five packets per second. For example, consider a path A-B-C-D-E-F. Even when the rate of a flow traversing this path increases by one packet per second, a transmission on the link C-D will contend with all the other four packet transmissions on the path.

- *Decrease phase:* On the other hand, when the feedback rate is smaller than the current rate, the sender performs the decrease phase (lines 14-15) merely adjusting its current rate to the feedback rate.

- *Maintain phase:* If the available rate $R$ lies within $(S, S + \phi * S)$, the sender maintains its rate. Thus, unlike in TCP which has to be either in the increase or decrease phases, ATP, given stable network conditions, can operate in a state of equilibrium. We demonstrate this behavior in Section 5.

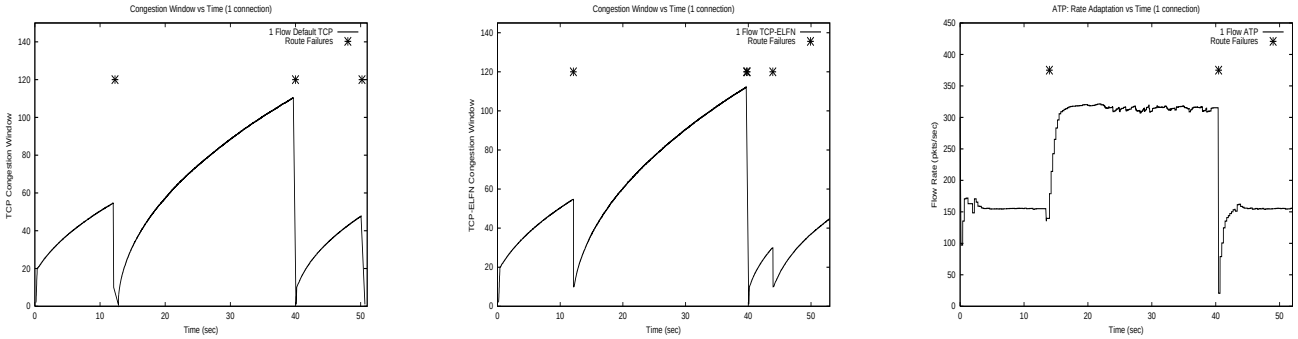
Note that the above congestion control decisions can be taken by the ATP sender only when it receives the rate feedback from the receiver correctly. It is possible that the rate feedback from the receiver is lost due to path failures on the reverse path. ATP addresses this issue by performing a multiplicative decrease of the sending rate for every epoch it does not receive feedback from the receiver, up to a maximum of two epochs. If it does not receive any feedback at the end of the third epoch, the ATP sender goes into its connection initiation phase, sending one probe every epoch till it hears back from the receiver. Note that once it hears from the receiver, it will use the rate on the feedback packet for its transmissions as part of its quick-start mechanism.

### 4.3.3 Reliability

As described in Section 4.2, the receiver as part of its periodic feedback sends information about any holes in the data stream it has received. The ATP sender treats the SACK information just as in TCP by maintaining a SACK scoreboard data structure. Data marked to be retransmitted are sent with a higher preference than new data. Note that the congestion control mechanism in ATP is decoupled from the reliability mechanism. Hence, while the congestion control protocol determines the rate at which the sender should be sending, the reliability mechanism ensures that packets queued for retransmission are sent preferentially when the send timer expires. In other words, the retransmissions are performed within the regular transmission rate determined by the congestion control algorithm.

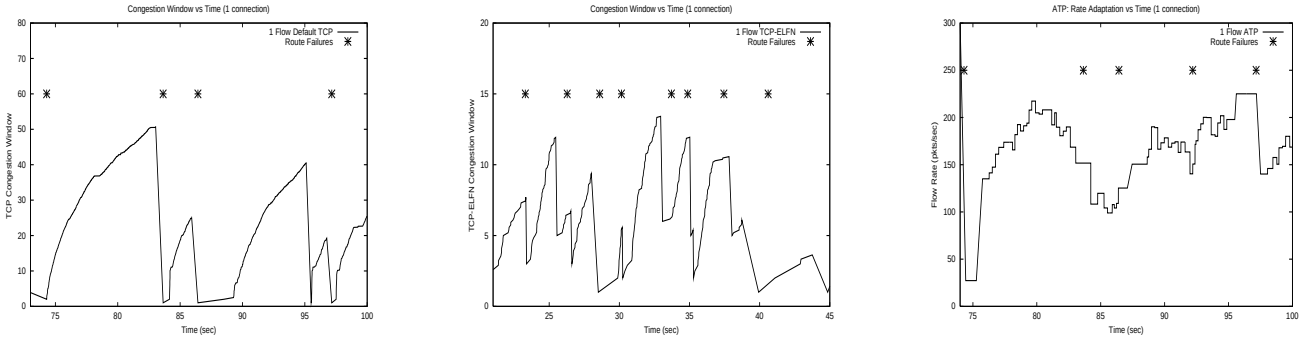
In addition to the receiver informing the sender about losses, when there is a path failure, the ATP sender uses explicit link failure notification from the appropriate intermediate node. When such feedback is received, the ATP sender immediately enters the connection initiation phase as part of its recovery mechanism after a route switch. In the connection initiation phase, the ATP sender, for every epoch, sends a probe packet to the receiver. The probe packet is piggybacked on the next in-sequence data packet queued for transmission. Even if there are suffix losses due to the path failure, the subsequent periodic probe packets sent by the sender serve to elicit a feedback packet from the receiver containing the appropriate SACK information.

Note that the ATP sender will typically receive the link-failure notification before (if at all) it receives SACK information from the receiver. This is because the SACK information from the receiver will be generated only when there are no suffix losses because of the path failure. However, suffix losses will be the norm unless some route salvaging is done by the intermediate nodes. Even if both the link-failure notification, and the receiver's periodic feedback packets are lost, the sender will eventually enter the probe phase due to lack of feedback from the receiver and hence recover from any losses.




(a) Default TCP (b) TCP-ELFN (c) ATP

**Figure 8: Congestion window/rate progression vs. time [1 Flow]**




(a) Default TCP (b) TCP-ELFN (c) ATP

**Figure 9: Congestion window/rate progression vs. time [25 Flows]**

# 5. PEFORMANCE EVALUATION

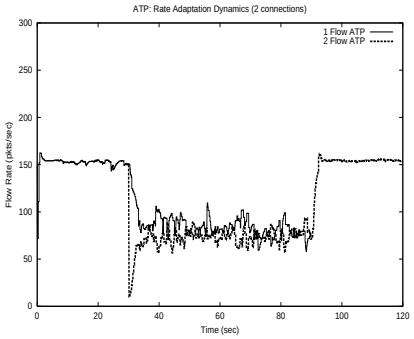
## 5.1 Environment

We use the network simulator *ns2* for all our simulations. The *setdest* tool in *ns2* is used to generate the random topologies for the simulations. The mobility model used for topology generation is the *random waypoint model*. All the simulations are performed for a 1000m $\times$ 1000m grid consisting of 100 nodes, distributed randomly over the two-dimensional grid. The source-destination pairs are randomly chosen from the set of 100 nodes in the network. We consider speeds of 1 m/s (pedestrian), 10 m/s and 20 m/s (vehicular) in our simulations. We also study the effect of load on the network by investigating scenarios with 1, 5 and 25 connections respectively. FTP is the application that we use over TCP for all the flows in the network, including those flows in the results that were presented in Section 2. The packets generated are of size 512 bytes in all the simulations. The performance of ATP is evaluated and compared against default TCP and TCP-ELFN for typical network scenarios outlined above. The metrics that we employ to measure the performance of the new transport protocol are instantaneous throughput, aggregate throughput and normalized standard deviation. By instantaneous throughput we refer to the congestion window progression for default TCP and TCP-ELFN, and the rate progression for ATP. The aggregate throughput is measured in kbps and reflects the number of packets successfully received at the destination. The normalized standard deviation measures the standard deviation between the individual flow throughputs in a particular scenario normalized to the average throughput for that scenario. Thus, the normalized standard deviation is a representative measure of the global fairness that is provided by the transport protocol. All the simulations are run for 100 seconds. Every flow in the network exists for the entire simulation run, and each data point on the graph is averaged over 10 simulation runs.

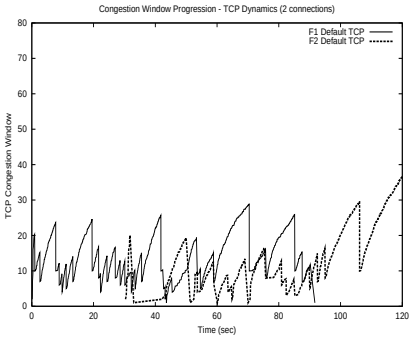
## 5.2 Results

### *5.2.1 Instantaneous Throughput*

We present snapshots of instantaneous throughput results for default TCP, TCP-ELFN and ATP for a single connection and a 25-connection scenario for a speed of 20m/s in Figures 8 and 9 respectively. The instants of route failures are also indicated on the plots. TCP invariably experiences a timeout and performs a slow-start even on a route failure since it does not distinguish between congestion losses and mobility losses. We have already demonstrated in Section 2 that around 90% of the time the routes recomputed on route failures are new ones, and hence TCP's multiplicative decrease is unwarranted. We make the following three key observations with respect to ATP's rate progression: (i) Since ATP does not use loss as an indicator of congestion, it does not decrease its rate on route failures unless dictated by its rate adaptation mechanism to do so. (ii) Owing to its quick-start mechanism of aggressively catching upto the available bandwidth, even when it experiences a route failure and is forced to decrease its rate, it is able to catch up to the available bandwidth on the recomputed route in a shorter time duration. (iii) Even though ATP aggressively tries to catch up to the available bandwidth, once it does reach the available capacity, it tries to maintain a steady rate without large fluctuations. Hence the rate adaptation mechanism stabilizes once the available capacity has been reached. This can be observed from Figure 8 (c) and Figure 9 (c).




(a) ATP rate adaptation




(b) TCP congestion window adaptation

**Figure 10: Instantaneous throughput dynamics [2 Flows]**

While the scenarios discussed thus far comprise of static traffic loads, to investigate the performance of ATP's rate adaptation mechanism in the event of network traffic dynamics, we consider a simple topology consisting of a five-hop linear chain with two flows *F1* and *F2*. *F1* exists between [0, 90]s, while *F2* exists between [30, 120]s. The instantaneous rates of the two flows are presented in Figure 10(a). Both flows achieve an aggregate average rate of about 100 pkts/s. As can be seen, after the arrival of *F2*, the two flows converge to the fair share of the available channel capacity almost instantaneously. Also when *F1* leaves, *F2* is able to catch up to the total available capacity in a short time. The corresponding instantaneous throughput dynamics for two TCP flows is shown in Figure 10(b). It can be seen that the convergence to the fair share for the TCP flows *F1* and *F2* is slow and oscillatory. Further when *F1* leaves, *F2* is unable to catch up to the available capacity quickly unlike in the case of ATP. This can be attributed to the fact that ATP uses explicit rate feedback, while TCP relies on its "blind" linear increase mechanism to ramp up to the available rate.

### *5.2.2 Aggregate Throughput*

For the single connection scenario, we observe in Figure 11(a) that for a typical network topology, the throughput achieved by ATP is almost 100% more than the protocols under comparison. TCP-ELFN is able to provide an improvement of around 20-25% over default TCP. Since ATP uses smarter start-up and rate based congestion control mechanisms, in a lightly loaded scenario as this one, it is able to utilize the underlying network bandwidth more efficiently. The appropriate reaction of ATP to mobility related losses prevents it from any performance degradation.

For multiple connections, we find that the aggregate throughput achieved by ATP is around 25-30% more than that of default TCP and TCP-ELFN for the case of 5 and 10 connections as shown in Figures 11(b) and (c), while it is about 10% for the 25 connections scenario as shown in Figure 11(c). The performance of TCP-ELFN degrades for the multiple connection scenarios and is less than that of default TCP. This has already been pointed out in earlier works [7]. From these results it can be observed that though the absolute performance of ATP is always better than that of default TCP and TCP-ELFN, its margin of performance improvement decreases as the load on the network increases. We attribute the following reason to this observation: As the load on the network increases, though some flows undergo performance degradation, there are other flows in the network that can potentially utilize the underlying bandwidth at the cost of the degrading flows. Thus the overall utilization of network capacity becomes better which in turn decreases the room for performance improvement for ATP. However, fairness will remain a concern even at higher loads. We present the fairness properties of ATP in Section 5.2.3.

### *5.2.3 Fairness*

In order to address the degree of global fairness provided by ATP in comparison to the protocols under consideration, we present the normalized standard deviation results in Figures 12(a), (b) and (c) for 5, 10 and 25 connections respectively. This metric can be thought of as being representative of an unfairness index. Hence the higher the normalized standard deviation the higher is the degree of unfairness in the network. It is evident from the results that ATP is able to decrease the degree of unfairness by as high as 40%. The reason being that when an intermediate node servicing several flows experiences congestion, it sends back feedback of congestion to all the sources of the flows being serviced by it. The sources respond in an identical manner to this congestion indication thereby leading to a higher degree of fairness in the network.

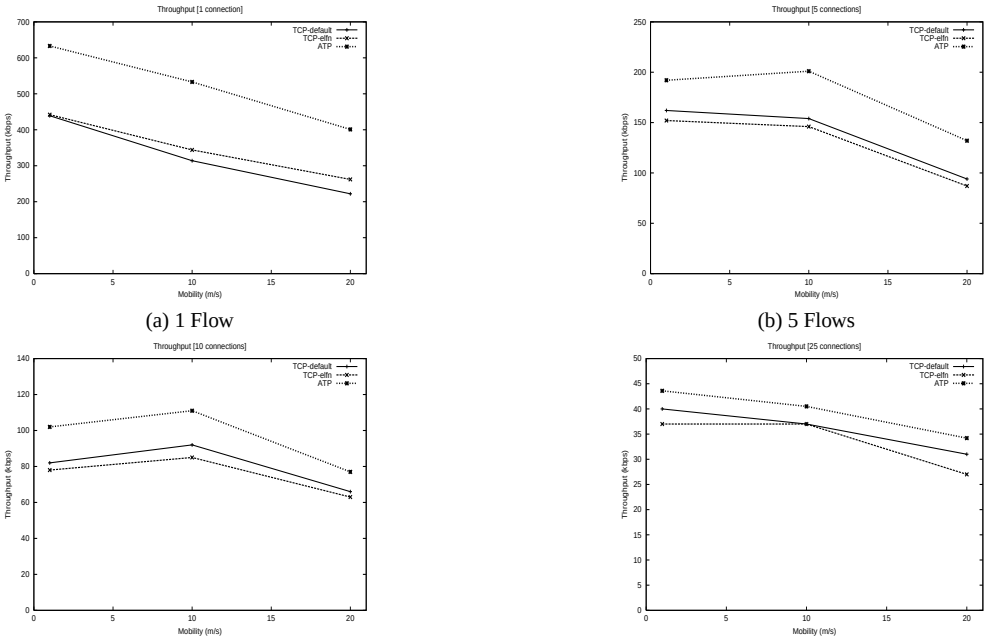
## 5.3 Summary

To summarize, we have shown the effectiveness of ATP's rate adaptation mechanism in this section. Further, we have evaluated the performance of the proposed transport protocol under different mobility and load conditions. The results clearly indicate the significant performance improvement that ATP provides over default TCP and TCP-ELFN. In addition ATP also achieves a higher degree of global fairness in the network.

# 6. RELATED WORK

TCP performance over cellular wireless packet-data networks has been the focus of research for a number of years, but recently, the focus has shifted to studying its performance on mobile ad-hoc networks. Proposals that address the problem of TCP performance over cellular wireless networks are typically of three types: (i) Improving the reliability at the link layer [16], (ii) Introducing TCP aware smarts at a central entity like the base station [12], and (iii) Split connection methods [17], which distinguish between the wireless and the wired domains in wireless networks. These approaches improve performance by keeping the TCP sender unaware of the loss characteristics of the wireless link, and thus preventing those from affecting the congestion control mechanisms of TCP. Such techniques cannot be deployed over ad-hoc networks because they require infrastructure support, and do not address the issue of losses due to route failures.

Several research works have attempted to identify factors affecting the performance in cellular and multi-hop wireless network sce-




(a) 1 Flow (b) 5 Flows

(c) 10 Flows (d) 25 Flows

**Figure 11: Throughput vs. Mobility**

narios [4, 5, 14]. [4, 14] study the effect of routing and link layer mechanisms on TCP performance. Cache management strategies are investigated and the effect of link layer re-transmissions on TCP throughput is discussed in [4]. However, the study is conducted on a static wireless network. While [14] comprehensively identifies the problems with TCP in dynamic multi-hop wireless networks, the mechanisms proposed at the media access and routing layers do not address all the issues.

Recent work [6–10] discusses the effect of mobility on TCP performance and suggests various transport layer mechanisms to solve the problems caused due to mobility. [6] evaluates an explicit link failure notification technique (ELFN) in the context of improving TCP performance over multi-hop mobile ad-hoc networks. They study the effect of link failures due to mobility on throughput and show through simulations that ELFN improves the performance of TCP. However, the focus is on a single connection and lightly loaded scenario. [7] also studies the performance of ELFN on static and dynamic networks and corroborates the results obtained in [6]. [8] discusses a mechanism called TCP-Feedback, which uses route failure and re-establishment notifications to provide feedback to TCP, and thus reduce the number of packet retransmissions and TCP back- offs during route calculation, to improve throughput. However, this mechanism is not evaluated in a mobile ad-hoc network environment. [9] studies the performance of TCP on three different routing protocols and proposes a heuristic called fixed RTO, which essentially freezes the TCP RTO value whenever there is a route loss. They also evaluate the effectiveness of TCP's selective and delayed acknowledgments in improving the performance. [10] provides a transport layer solution to improving TCP performance. It introduces a thin layer between the transport and underlying routing layers, which puts TCP into persist mode whenever the network gets disconnected or there are packet losses due to high bit error rate. Thus, this thin layer acts as a shield to TCP, protecting it from the underlying behavior of an ad-hoc network.

In a different context, the NACK-oriented reliable multicast protocol (NORM) [18] provides NACK based reliability and is similar to ATP since it employs rate based transmissions. However the other fundamental differences that exist between TCP and ATP in terms of the start-up behavior, congestion control and reliability mechanisms, still hold true for NORM and ATP.

Finally, while several approaches have been proposed to leverage multi-path routing in ad-hoc networks [19, 20], ATP is designed for single path connections. Note that TCP is also designed only for single-path connections and will also suffer drastically if multi-path routing. However, a transport layer framework such as [21] can be used in tandem with ATP to effectively support connections traversing multiple paths.

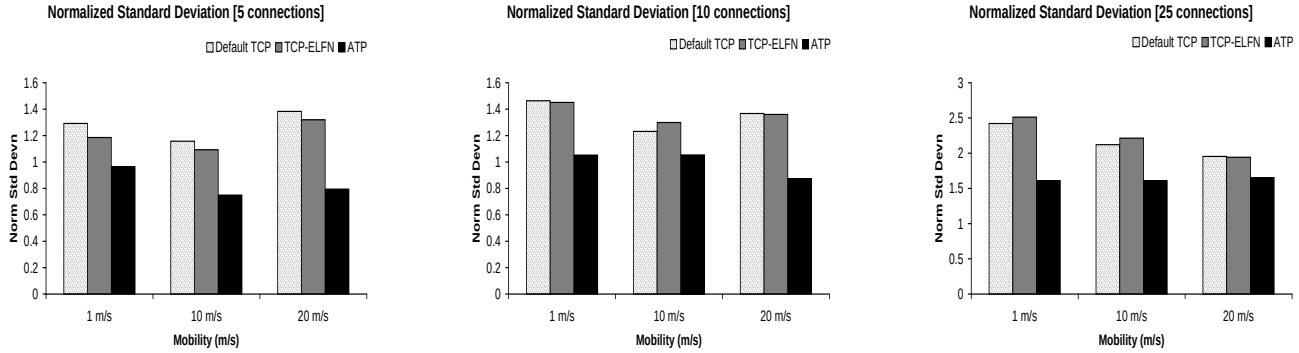
In summary, most of the related research aims at either augmenting TCP with some mechanism or protecting TCP from the nuances of the ad-hoc network, so as to improve performance. ATP on the other hand is not a TCP-variant and is tailored specifically to suit the characteristics of ad-hoc networks.

# 7. CONCLUSIONS

The behaviour of TCP over ad-hoc networks is studied extensively in this paper. We infer from the results that a majority of the components of TCP are not suitable for the characteristics of ad-hoc networks. Various reasons are discussed, and the insights gained from the study are used to motivate a new transport protocol called ATP, which is better suited for ad-hoc networks. The protocol addresses all the problems that TCP faces when deployed over ad-hoc networks, and thus shows considerable performance improvement over TCP and TCP-ELFN.

# 8. REFERENCES

[1] D. Johnson, D.A. Maltz, and J. Broch, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks ," in *MANET Working Group. IETF, Internet Draft, draft-ietf-manet-dsr-07.txt*, Feb 2002.




(a) 5 Flows (b) 10 Flows (c) 25 Flows

**Figure 12: Normalized standard deviation**


[2] C. E. Perkins and E. M. Royer, "Ad-hoc On-demand Distance Vector (AODV) Routing," in *MANET Working Group. IETF, Internet Draft, draft-ietf-manet-aodv-12.txt*, Nov 2002.

[3] B. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving Performance of TCP over Wireless Networks," in *Proceedings of 17th International Conf. on Distributed Computing Systems (ICDCS)*, Baltimore, MD, May 1997.

[4] G. Holland and N. H. Vaidya, "Impact of Routing and Link layers on TCP Performance in Mobile Ad-hoc Networks," in *Proceedings of IEEE WCNC*, New Orleans, September 1999.

[5] M. Gerla, K. Tang, and R. Bagrodia, "TCP Performance in Wireless Multi Hop Networks," in *Proceedings of IEEE WMSCA*, New Orleans, Feb 1999.

[6] G. Holland and N. H. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," in *Proceedings of ACM MOBICOM*, Seattle, WA, Aug 1999, pp. 219–230.

[7] J. P. Monks, P. Sinha, and V. Bharghavan, "Limitations of TCP-ELFN for Ad hoc Networks," in *Workshop on Mobile and Multimedia Communication*, Marina del Rey, CA, Oct. 2000.

[8] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback Based Scheme for Improving TCP Performance in Ad-Hoc Wireless Networks," in *Proceedings of International Conference on Distributed Computing Systems*, Amsterdam, May 1998, pp. 472–479.

[9] T. D. Dyer and R. Bopanna, "A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks ," in *Proceedings of ACM MOBIHOC 2001*, Long Beach, CA, Oct 2001.

[10] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," in *IEEE Journal on Selected Areas in Communications*, 2001.

[11] C. E. Koksal and H. Balakrishnan, "An Analysis of Short-term Fairness in Wireless Media Access Protocols (poster)," in *Proceedings of ACM SIGMETRICS, Measurement and Modeling of Computer Systems*, Santa Clara, CA, 2000, pp. 118–119.

[12] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, "Improving TCP/IP performance over wireless networks," in *Proceedings of ACM MOBICOM*, Berkeley, CA, Nov. 1995.

[13] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks," in *Proceedings of ACM MOBICOM*, Seattle, WA, Aug. 1999.

[14] V. Anantharaman and R. Sivakumar, "A Microscopic Analysis of TCP Performance Analysis over Wireless Ad Hoc Networks," in *Proceedings of ACM SIGMETRICS 2002. (Poster Paper)*, Marina del Rey, CA, June 2002.

[15] T. Henderson and R. Katz, "Satellite Transport Protocol (STP): An SSCOP-based Transport Protocol for Datagram Satellite Networks," in *Proceedings of 2nd Workshop on Satellite-Based Information Systems (WOSBIS)*, Budapest, Hungary, 1997.

[16] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP Performance over Wireless Networks at the Link Layer," *Mobile Networks and Applications*, vol. 5, no. 1, pp. 57–71, 2000.

[17] A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," in *Proceedings of 15th International Conference on Distributed Computing Systems (ICDCS)*, Vancouver, British Columbia, Canada, May 1995.

[18] M. Handley, C. Bormann, B. Adamson, and J. Macker, "NACK Oriented Reliable Multicast (NORM) Protocol Building Blocks," in *Internet Draft, RMT Working Group, draft-ietf-rmt-bb-norm-05.txt*, March 2003.

[19] A. Nasipuri and S. Das, "On-Demand Multipath Routing for Mobile Ad Hoc Networks," in *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN), Boston, MA, October 1999, pp. 64-70.*, 1999.

[20] M. K. Marina and S. R. Das, "On-demand Multipath Distance Vector Routing in Ad Hoc Networks," in *Proceedings of IEEE ICNP*, Riverside, CA, 2001.

[21] H.-Y. Hsieh and R. Sivakumar, "A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homed Mobile Hosts," in *Proceedings of ACM MOBICOM*, Atlanta, GA USA, September 2002.