An Efficient Transport Protocol for Ad Hoc Networks: An End-to-End Freeze TCP with Timestamps

Sung Rae Cho, Harsha Sirisena, and Krzysztof Pawlikowski

Abstract: **In ad hoc networks, loss-based congestion window progression by the traditional means of duplicate ACKs and timeouts causes high network buffer utilization due to large bursts of data, thereby degrading network bandwidth utilization. Moreover, network-oriented feedbacks to handle route disconnection events may impair packet forwarding capability by adding to MAC layer congestion and also dissipate considerable network resources at reluctant intermediate nodes. Here, we propose a new TCP scheme that does not require the participation of intermediate nodes. It is a purely end-to-end scheme using TCP timestamps to deduce link conditions. It also eliminates spurious reductions of the transmission window in cases of timeouts and fast retransmits. The scheme incorporates a receiver-oriented rate controller (*rater*), and a *congestion window delimiter* for the 802.11 MAC protocol. In addition, the transient nature of medium availability due to medium contention during the connection time is addressed by a freezing timer (*freezer*) at the receiver, which freezes the sender whenever heavy contention is perceived. Finally, the sender-end is modified to comply with the receiver-end enhancements, as an optional deployment. Simulation studies show that our modification of TCP for ad hoc networks offers outstanding performance in terms of goodput, as well as throughput.**

Index Terms: **End-to-end rate control, Freeze TCP, receiver advertised window, TCP timestamps, 802.11 MAC.**

I. INTRODUCTION

The services offered by 3G wireless networks to mobile users will create a huge volume of network traffic. This increasing trend in network traffic will continue in the future due to an ever increasing number of mobile users. Also recent developments in mobile technology have sped up adoption of various new functionalities, such as full multimedia services requiring high bandwidth and high data transmission rates. Furthermore there is a rapid progression towards an all IP based 4G networks, which can offer global connectivity between end users. Presently mobile devices in use include cellular phones, PDAs, and laptops equipped with radio transceivers. All of them allow mobile users to communicate with each other in portable environments of heterogeneous networks. However, such mobile and portable devices rely on existing network infrastructures, such as routers, base stations, and access gateways, for seamlessly connecting with each other. The growing demand for mobile computing devices promotes ubiquitous networking that does not require a pre-existing infrastructure. Such standalone networks would be implemented as mobile wireless ad hoc networks, MANETs [1].

Manuscript received July 31, 2004.

S. R. Cho is with the Electronics and Telecommunications Research Institute, Daejeon, Korea, email: srcho@etri.re.kr.

H. Sirisena and K. Powlikowski are with the University of Canterbury, New Zealand, email: sirisehr@elec.canterbury.ac.nz, k.pawlikowski@ieee.org.

This is a challenging research area, with results published in specialized technical journals and conferences. The following paragraphs detail several important peculiarities of MANETs which require special consideration.

Bandwidth-limited shared medium–Mobile nodes share a common medium to communicate with each other on a random access basis. As a result of this, MAC collisions might be frequent in the case of highly congested environments. In addition, many other factors, such as physical obstacles, noise, and interference with other nodes, and vulnerable link sustainability, can drastically restrict the available raw bandwidth. Notice that the available bandwidth is highly variable during the connection time, and it usually never fully utilizes the raw bandwidth.

Power-limited mobile nodes–The processing capability of mobile nodes is restricted because of limited battery power. This constraint requires low processing overheads at nodes. Inter-router exchange and network-based notification of control packet should thus be highly avoided. In particular, if dedicated routers do not implement a flooding control algorithm and QoS mechanisms are not available, intermediate routers might not want to consume considerable power for serving others' applications. In this sense, network-originated control beacons required for certain applications and session-related states maintained in the network are not desirable. Hence there is a need for end-to-end based TCP (transmission control protocol) mechanisms in MANETs.

Network partitioning–Since there are no central base-stations in ad hoc networks, it is likely that a part of a network becomes entirely partitioned from the rest of the network. If a sender and a receiver of a TCP connection lie in different partitions, packets transmitted get dropped by the network, resulting in invocation of the congestion control algorithm. Then, successive failures (timeouts) in the slow start phase will shrink the slow start threshold ($ssthresh$) unnecessarily and, in turn, underutilize the available network resources. Currently, no explicit end-to-end scheme is available for restoring the unnecessarily lowered $ssthresh$—for instance, ATP [2] requires network-originated feedbacks for fast restoration. TCP-Westwood [3] mitigates the impact of spuriously reduced $ssthresh$ by means of end-to-end bandwidth estimation, but this cannot be directly employed in ad hoc networks.

High bit error rate–The bit error rate in an error-prone wireless links is commonly in the order of 10^{-6} or even worse. Thus, each node definitely requires reliable link layer protocol(s) to assure delivery of MAC frame across links, which then improves the end-to-end throughput. However, use of a link layer scheme can cause inflation of the round trip time of packets and then lead to spurious timeouts. Floyd and Ludwig introduced DSACK [4] and Eifel algorithm [5], respectively, to address this problem.

These algorithms can, in view of the modified transport layer, mitigate the impact of spurious timeouts by undoing the reductions made. The support of such complementary algorithms can further encourage deployment of more reliable link layer protocol(s) for avoiding per-hop link errors as much as possible.

Multipath routing and frequent route changes–Multiple path routing is commonly utilized in ad hoc networks, although this can degrade TCP performance because a TCP receiver considers the sequential order of packets for ultimately sizing the congestion window. For instance, such routing protocol as TORA (temporally-ordered routing algorithm) maintains multiple routes between source-destination pairs, in order to minimize frequent route recomputation. However, it might result in a large number of out-of-sequence packets arriving at the receiver. In response, the receiver duplicates ACKs of out-of-sequence packets. This causes the TCP sender to invoke the congestion control algorithm unnecessarily. Spurious reductions made in the congestion window and $ssthresh$ should be avoided. As a matter of fact, the sender might not completely prevent spurious fast retransmits due to the inevitably high out-of-order delivery rate in terms of dynamic ad hoc routing. The Eifel algorithm, TULIP [6], and D-SACK endeavor either to alleviate or avoid the impact of spurious fast retransmits to a certain extent, although it can be at a considerable computational expense. Namely, the congestion window has to be recomputed or properly set with an appropriate $ssthresh$ after route reconnection; otherwise, it is likely to either overwhelm or underutilize the network capacity. For instance, a fast increase of the congestion window, resulting from a high setting of $ssthresh$, can be harmful because it might hamper transmissions from adjacent nodes, causing high buffer queue levels. In brief, TCP needs to be aware of path rerouting for the congestion window to be set appropriately, according to the rerouted path condition.

With this in mind, the proposed TCP scheme modifies the receiver to enable it to estimate $ssthresh$ representing the current medium contention degree of the path, and the sender to consider the $ssthresh$ for the new rerouted path with the aim of eliminating the impact of spurious fast retransmits. That is, in our proposal, the receiver informs the sender of an estimated $ssthresh$ periodically, and the sender does not shrink the congestion window drastically but only to the estimated threshold. In addition, such solution can be useful in the case of timeouts that can quench $ssthresh$ unnecessarily, too.

Current TCP schemes devised for either wireline or wireless networks cannot be directly employed in ad hoc networks. The wireline TCP has been optimized for buffer related packet losses that dominate in fixed wireline networks, so the TCP sender quenches its transmission window drastically each time a packet loss occurs. However, ad hoc links are more likely to suffer from link-related errors than from buffer overflow [7]. Therefore single wireless link related errors need not shrink the transmission rate drastically unless burst wireless errors occur. Most wireless TCP schemes use a central base station which supervises connections made between end nodes lying in two different networks (as the *split* TCP connection[1] does). Therefore, neither of these two types of TCP can be used without a proper modification.

Most TCP schemes proposed for ad hoc networks deal with route disconnection problems, by placing the underlying TCP mechanism into the persist state (*frozen state*) until a new link is restored, by way of network-supported signaling (e.g., ICMP, ELFN, etc.). In ad hoc networks, the dynamic nature of network topology requires to understand the complex interactions between TCP and lower layer protocols. As a matter of fact, vertical information flow required between layers violates the standalone modularity of TCP for conventional flows that use only strict peer-to-peer horizontal communication between layered protocols. However, flexible inter-layer information flow with lower layer protocols can provide useful knowledge of network link conditions in so far as the routing protocol plays a role in sustaining a path between the end TCP entities, and the MAC protocol is responsible for per-hop basis bandwidth provision. For instance, Sun and Man [12] proposed an enhanced inter-layer control mechanism (ENIC) for ad hoc networks that enables layer-to-layer vertical communication among the layered protocols employed, where upper-layer protocols bind more closely with lower-layers, leading to more efficient and direct inter-layer operations. This can drastically reduce the overhead of layer-specific control messages. Goff *et al.* [13] introduced Freeze TCP, which however cannot be directly employed in ad hoc networks, for handoffs to initiate freezing of the sender's TCP window in response to impending link disconnection determined by lower layer protocol at the receiver. Freeze TCP does not require modifications of the sender but requires prediction of impending disconnection of the last hop wireless link determined by the MAC protocol (TCP-aware link layer scheme). We therefore assert that an interoperating mechanism between layered protocols is justified to determine dynamic bandwidth availability. There is no doubt that, without freezing, the TCP sender will suffer from a number of unnecessary timeouts, resulting in a waste of network resources (e.g., a spuriously diminished $ssthresh$ [3] as well as congestion window, and inconsistent RTO set after a new route is restored).

The explicit link failure notification (ELFN) approach [14], [15], based on inter-layer information delivery from the MAC protocol, is addressed as a typical solution to the route disconnection problem. It is a flexible and sustainable remedy for such problems, although some argue that it exhibits sub-optimal behaviors [15]. It uses ELFN occasionally originated by a router whose next hop link was broken, so that the TCP end host can take the information and stall transmission until the link gets restored. As a drawback, the sender and receiver rely upon the performance of all intermediate routers along the path, where routers propagate ELFN messages according to link viability. Also the sender must be modified to function with the ELFN messages.

ATCP [16] also uses two kinds of explicit control beacons, such as ECN (explicit congestion notification) and ICMP, in order to control flow rate (ECN messages inform about network congestion, distinguishing it from instant wireless link corruptions, and ICMP "destination unreachable" messages that the

[1]The *split* connection (e.g., I-TCP [8] and M-TCP [9]) or proxy approach (e.g., Snoop [10] and WTCP [11] as a TCP aware link layer scheme) requires an intelligent base station between end hosts, thereby is unacceptable for ad hoc networks because these networks have no fixed base station.

network is partitioned, or that no route exists). In fact, the sender's behavior in terms of its flow rate control mechanism should not be fully dependent upon receipt of such messages because (i) ICMP messages are transmitted via the UDP protocol that does not guarantee their deliveries and (ii) ECN-designated packets may be lost over wireless links. The loss of an ICMP message results in a number of retransmit timeouts followed by successive retransmissions since route disconnection periods would have occurred. ECN, useful for indicating congestion, cannot eliminate packet losses completely, and the end nodes should be able to interpret packet losses as indications of wireless medium losses (corruptions due to high BER) instead of considering them as the effects of congestion only. In addition, all the routers along the path must be ECN-enabled, and the ICMP messages originated from the network might cause network links to be congested due to the considerable bandwidth consumed.

In fact, such explicit network feedbacks are more accurate and fast since intermediate routers can detect failures sooner. However, they can result in considerable network bandwidth consumption by intermediate routers. So, one might not want to use them if they do not result in a reasonable performance improvement.

The state-of-art TCP schemes proposed may be classified into either network detection-based or end node detection-based. For example, TCP-Feedback [17], ELFN-based [14], ATCP [16], TCP-Bus [18], TCP-EXACT [19], and ATP [2] rely upon network signaling for detecting path anomalies. So, they are related to the network detection strategy and dominated by the performance of network detection mechanism. This implies that the reliability of the detection strongly influences the performance of end-to-end flow control. TCP-EXACT [19] and ATP [2] rely on network originated information to control the sender's flow rate, and thus its delivery timeliness, as well as considerable bandwidth expense, matters. On the other hand, TCP-DOOR [20], Fixed RTO [21], ADTCP [22], and Edge-based TCP [23] are related to end node detection without support of any intermediates in the network, keeping the end-to-end TCP semantics in flow control. As a typical example of the end-to-end approach in terms of frequent route change problem, Wang and Zhang [20] introduced a purely end-to-end based scheme (TCP-DOOR), so that either sender or receiver works without any network based control beaconing, in order to detect whether the route has been changed. ADTCP cited in [22], likewise, uses an end node determination mechanism so that the receiver-end categorizes the network condition with several metrics: IDD (inter-packet delay difference), STT (short-term throughput), PLR (packet loss rate), and POR (packet out-or-order delivery rate). Then a decision is made with help of decision inference algorithms used for determining a threshold-based and a maximum likelihood classifier, with inputs from the metrics measured when packets are absent in arriving sequences. In the meantime, the receiver is able to determine whether the network link has gone into any of abnormal stages, such as congestion, router change, channel error, or route disconnection stage. Then, the sender responds accordingly. However, this end-to-end approach does not involve an explicit window control during in-order deliveries of packets. It only monitors by inferring the causes of packet losses, out-of-sequence packet deliveries. Such end-to-end approaches require special-purpose modifications at one end or both. Hence, they might neither guarantee backward compatibility nor its transparent, robust functionality when they interoperate with an unmodified counterpart in heterogeneous networks, and could eventually end up with sub-optimal behaviors.

A few TCP schemes [2], [19], [24] attempt to coordinate the transmission rate according to the bandwidth available under lower layer protocols. Since mobile nodes compete with each other for a single common channel, the bandwidth available per node fluctuates during communication. In case of intensive medium contention occurring in the middle of the path, the network-supported (inter-layer based) scheme can be one of solutions to mitigate the degree of the medium contention. It will freeze the sender instantly or reduce further transmissions that deteriorate the status of spatial channel use, in terms of the well-known MAC terminal problems. Ahn *et al.* [24] introduced rate and admission controllers (SWAN) functioning at the sender-end by monitoring incoming packet delays, taking into account traffic types (i.e., best effort traffic and real-time traffic) and operating without any network feedbacks. The SWAN AIMD rate control mechanism is capable of maintaining the MAC delay under a certain target boundary. However, it does not provide a complete TCP flow control scheme. To the best authors' knowledge, no complete end-to-end TCP scheme, to coordinate the transmission rate according to the link capacity available under lower layer protocols, has been proposed. We investigate a purely end-to-end approach in order to control bandwidth-constrained links in ad hoc networks without explicit signaling from the network.

The rest of the paper is organized as follows. Deficiency of the normal TCP, using TCP SACK as the baseline, is discussed in Section II. Section III explains an instantaneous rate computation method (Section III-B) and a window advertisement (Section III-C) according to the forward link delay of data packets. Then, we introduce an explicit freezing signal (ZWA), which is invoked when a heavy medium contention is perceived at the receiver (Section III-D). In addition, a modified congestion control algorithm for ad hoc networks (which introduces *ad hoc* sender enhancements optimally complying with the receiver-end enhancements) as well as an add-on probing mechanism (superseding the retransmission timeout (RTO) backoffs of TCP), are proposed in Section III-E. Finally, performance of our proposal is assessed on the basis of simulation results presented in Section IV. This is followed by Section V that concludes with a brief overview of future research issues.

II. INEFFICIENCY OF THE BASELINE

The reactive congestion window progression of the normal TCP (e.g., TCP SACK as the baseline) has been optimized for buffer related packet loss over fixed wireline networks, i.e., the TCP sender quenches the transmission window each time when a packet drop is perceived. On the other hand, ad hoc networks are not likely to suffer from buffer overflow but rather from link-related errors [7]: Such as collisions in the MAC layer, wireless link corruptions, and routing failures. Therefore single wireless link related errors need not shrink the transmission window

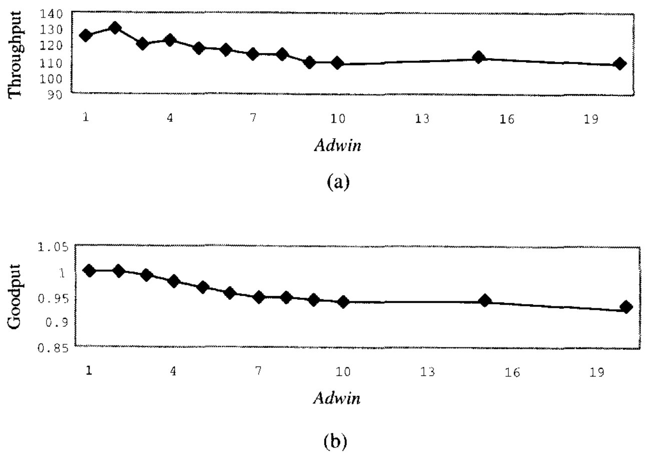


Fig. 1. Throughput (kbps) and normalized Goodput for TCP SACK with different advertised windows in a 5-hop stationary string topology (active for 100 sec with 1000 byte packets): (a) Throughput according to a limiting $adwin$, (b) goodput according to a limiting $adwin$.

drastically unless burst wireless errors occur. In other words, the reactive adjustment of the congestion window does not take into account the link condition (e.g., high BER, medium contention, and route disconnection) causing a packet loss. That is, it is impossible for the original TCP to classify the cause of a packet loss and perform reasonably by the traditional means of RTO and duplicate ACK.

Fig. 1 shows TCP throughput and goodput gains by limiting advertised window ($adwin$). Throughput is best at $adwin$ of 2, being 18.3% greater than at $adwin$ of 20 (at which the sender is solely *congestion window limited* for the whole connection time of 100 sec), and goodput by over 5%. Despite there being no competing traffic sources, the TCP sender suffered from a number of fast retransmits and timeouts. The results indicate that the dynamic adjustment of $adwin$ according to the degree of present medium contention can improve TCP throughput. Besides, if other traffic sources are present, throughput and goodput can also be improved by a suitable limiting $adwin$.

The study in [7] and [25] pointed out that there exists an optimal value for the TCP congestion window that maximizes TCP throughput. Fu *et al.* [7] argue the need for a MAC-aware congestion control mechanism according to MAC related delay and routing errors, because packet losses are almost always caused by medium contention. However, in view of the deficiencies of the 802.11 MAC protocol, e.g., well-known terminal problems, it would be hard to determine optimal congestion windows for randomly changing topologies.

Awareness of an optimum window size encourages investigating the optimum window according to network link condition (i.e., available maximum network *pipe*) so that the sender can lower network buffer utilization and thereby prevent unnecessary MAC collisions (i.e., collisions in the MAC layer due to well-known terminal problems along the path) to a considerable extent. With an increasing number of backoffs or local retransmissions due to high contention, or to other reasons such as link corruptions or routing failures, the receiver can limit the sender's transmission window, and so prevent queues building up along the path, thereby mitigating the level of the medium contention.

First, the following section explains an instantaneous rate computation method to give a window advertisement according to the forward link delay of data packets, and then introduces an explicit freezing signal (ZWA) that is invoked when heavy medium contention is perceived at the receiver. Finally, the *ad hoc* sender enhancements that optimally comply with the receiver-end enhancements (i.e., modified congestion control algorithm for ad hoc networks), as well as implement an add-on probing mechanism, superseding the retransmission timeout (RTO) backoffs of TCP, are proposed.

III. THE PROPOSED SCHEME

This section details the rationale of the newly introduced a purely end-to-end TCP scheme in which the receiver-end estimates the available bandwidth to an extent, without any network-originated signaling. Our newly proposed *ad hoc TCP* does not try to identify the cause of packet loss [22] but rather to adjust $ssthresh$ and restrict the congestion window, hopefully in advance of an actual packet drop, more frequently and meaningfully according to network condition changes (rather than just to shrink the congestion window drastically). To do so, the receiver informs the sender of an estimated $ssthresh$ that would indicate the available bandwidth and the sender controls the congestion window, as well as its $ssthresh$, when it encounters timeouts or fast retransmits. At other times, a drastic network condition change is notified by freezing zero window advertisement (ZWA).

A. Requirement

For the receiver to perceive the path condition, at least, two prerequisites are required:

Clock synchronization–Due to the small geographical areas covered by ad hoc networks, it should be possible for nodes to have synchronized configurations.[2] All nodes within the ad hoc network should have synchronized clocks to give meaningful information about the forward link delay (FLD) of data packets, and then:

- To compute maximally allowable bound capacities determined by the FLD (difference between the departure and the arrival times of packet time-stamped is measured at the receiver), and
- to find the worst link condition determined by FLD and its deviation (measured at the receiver), which are used for deciding when to propagate a feedback to freeze the sender.

Inter-layer information flow–TCP should obtain TTL from the IP header to determine the current hop length (n) between the end nodes. The hop length of a route may be updated (or maintained if the hop length is the same as previous; the inspection of the TTL value in each received data packet can not signify all route switches) every time the route has been partially or entirely rerouted.

[2]Specific procedures (activated passively during the three way handshake, or actively negotiated) could provide the participating mobile nodes useful information about, for example, the forwarding capacity allowed by present power levels, and allow passive coarse clock-synchronization mechanism and passive RTT measurements.

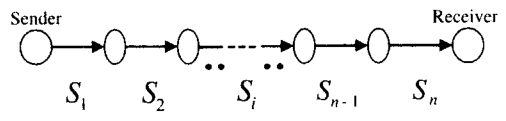


Fig. 2. Hop-based available throughput (bytes/s).

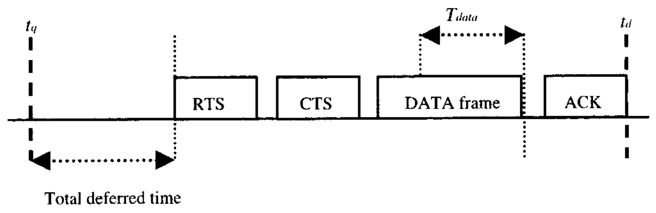


Fig. 3. Factors influencing the instantaneous throughout at a router, where t_q = the time queued at a router, and t_d = time drained out of the router.

B. Instantaneous Throughput Estimation (Rater)

The employed 802.11 MAC protocol plays a role in sharing the common channel for all competing nodes. It is thus a major factor in determining bandwidth availability at a particular node at a particular instant.

Fig. 2 shows an n-hop link between a source-destination pair, the instantaneously measured available throughput of each hop link being denoted by $S_1, S_2, \cdots, S_n$ as in (1).

Each S_i determines the time taken to forward a packet at that instant over a particular link, namely, the *instantaneous per-node delay*. That is, the amount of time spent in servicing a packet at a router (per-packet delay), mostly affected by neighboring medium competitors and queues building up.

$$\begin{aligned} S_{instantaneous} &= \frac{P}{t_d - t_q} \\ &= \frac{P}{MAC\_related\_delay + P/B_{raw}}, \end{aligned} \quad (1)$$

where P is a packet size in bytes, and the meanings of t_q (the time queued at a router) and t_d (the time drained out of the router) are explained in Fig. 3.

Queuing delay corresponds to the total time needed for dequeuing of packets already queued at a given router when a given packet arrives there. The time spent on queuing a given packet at a router is dominated by the *medium contention delay* and depends on the network load. It is very hard to determine it precisely in the case of mobile users who use diverse mobile computing devices. *Transmission delay* for a dequeued IP datagram is characterized by the line rate (nominal raw link capacity, denoted as B_{raw}) and the packet size, assuming that the *processing delay* related to *packetization* between layers is negligible. Each packet experiences the same *propagation delay* independently of the packet size.

MAC related delay, see (1) and Fig. 3, is used to compute the instantaneous throughput. This delay includes queuing, MAC contention delay, and propagation delays. However, the individual delays cannot be explicitly determined solely by the TCP receiver, except the transmission delays. MAC contention is the most dominant fluctuating factor in MAC related delays.

On the other hand, ATP [2] and TCP-EXACT [24], based on per-node and per-flow computations, respectively, require the involvement of intermediate routers to inform the sender about the available bandwidth whenever a packet is transmitted through them. However, the availability changes with time due to nodes' mobility and their interdependency, as well as unpredictable transient interventions (e.g., short bursts). Such dynamic interdependency between mobile nodes means that the resultant MAC related delay observed at the receiver changes with time, fluctuating with the level of medium contention. Thus, each hop link throughput is a transient quantity, and so can be validated only instantaneously. Thus, such router involvements might not lead to reasonable improvements because there is no guarantee of timely delivery of valid information, in spite of the considerable complexity of the schemes. As an alternative, we use the average bottleneck throughput, $S_{average}$, see (2), rather than an actual bottleneck throughput explicitly available from a router. Note that $S_{average}$ is the harmonic mean of $S_1, S_2, \cdots, S_n$. It is always greater than, or equal to, the actual bottleneck throughput; see (3).

$$FLD = \frac{P}{S_1} + \frac{P}{S_2} + \cdots + \frac{P}{S_n} = n \times \frac{P}{S_{average}}, \quad (2)$$

where

$$\begin{aligned} n\frac{P}{B_{raw}} < n\frac{P}{S_{\max}} &\le \left(FLD = n\frac{P}{S_{average}}\right) \\ &\le n\frac{P}{S_{bottleneck}} \le n\frac{P}{S_{\min}}. \end{aligned}$$

Thus,

$$S_{\min} \le S_{bottleneck} \le S_{average} \le S_{\max}. \quad (3)$$

In addition, allowing for variation with time, the instantaneous hop link throughput along the path can be delimited between two extreme values, $S_{\max}$ and $S_{\min}$, which bound the possible range of $S_{average}$, depending on the hop length between the end nodes, packet sizes, and the raw line rate (B_{raw}). A per-hop delay is acceptable only when the throughput is between $S_{\max}$ and $S_{\min}$. $S_{\max}$ defines a maximally allowable hop link throughput as a path-specific *reference metric*. This entity in particular delimits $S_{average}$, which depends on FLD and the current hop length. If $S_{average}$ exceeded $S_{\max}$, it would the most likely mean that the receiver has communicated with a wireline counterpart ($S_{average}$ might exceed $S_{\max}$ because a wireline hop link[3] throughput is much higher than an ad hoc wireless link one), or necessary minimum MAC overheads has been reduced somehow. If instantaneously measured $S_{average}$ is greater than $S_{\max}$, the receiver could suppose the presence of faster link(s) between end correspondents. Furthermore, $S_{average}$ could be employed as a metric for admission control of a real-time transport protocol because this metric is supposed to determine a hop-based capacity independent of RTT between flows (unfairness might occur due to the difference in $RTTs$ of long and short flows). $S_{\min}$ defines the minimal required hop link capacity. If the receiver detects that the rate adjusted goes

[3]Providing that a reasonable buffer size is used.

below $S_{\min}$, it might expect that a current hop link is going to be (or seems to be) disconnected, because too low throughput implicitly means high MAC related delay, and significant jitter effects at packet arrival times (data or ACK can be starving in either forward or reverse way). We use $S_{\min}$ for incorporating the likelihood of route breakage, since $S_{\min}$ itself can not be directly related with an imminent route breakage. In terms of bandwidth constraint 802.11 MAC characteristics, the freezing timer can determine $S_{\min}$ (i.e., the reciprocal of the freezing timer multiplied by MSS) because allowable traveling delay prior to the TCP sender's timeout could specify $S_{\min}$ in the sense that the freezing timer determines the least link capacity in terms of falsely induced RTOs at the TCP sender.

C. Congestion Window Delimiter

TCP basically uses a window based congestion control mechanism: Effectively the estimated rate (here it is $S_{bottleneck}$) is multiplied by the RTT in order to compute the *bandwidth-delay product*. That is, the sender injects this number of packets into the network *pipe*. However, due to the half duplex nature of the IEEE 802.11 specific characteristics, we use a minimum FLD ever observed in order to dimension *the length of the pipe*.

The initial receiver's advertised window ($adwin$) can be therefore as below:

$$\begin{aligned} Adwin_0 &\cong \frac{S_{average}}{P} \times RTT_{\min} \\ &= \frac{n}{FLD_0} \times RTT_{\min} \\ &\cong \frac{n}{FLD_0} \times 2 \times FLD_0 \\ &= 2n \quad \text{(segments)}, \end{aligned}$$

where FLD_0 is the initial FLD and $RTT_{\min}$ is a minimum RTT ever observed.

The initial $adwin$, valued $2n$, implies that we used the rate computed by per-node based packet forwarding time and that the sender puts $2n$ packets in the network pipeline, because of equal share of $2n$ hops in RTT of a packet. However, due to the limitation of single half duplex channel, each link can either transmit or receive only one packet at a time, and thus the network pipe cannot hold even n packets to be relayed concurrently over n-hop path. Neighboring nodes even in equal-distant string topology, where one node can only transmit to one-hop away node and might interfere with up to two-hop distant nodes, can interrupt the packet transmission at least in two-hop distant neighboring nodes. At each hop, one packet can be transmitted by a node, and at the same time another received; otherwise, queue builds up and RTT subsequently increases. Mathematical approaches to analysis of these situations can be found in [13] to evaluate the allowable maximum explicit windows according to the number of hops between end nodes, under the assumption that the forward link is identical to the reverse link. Simply stated, the 802.11 protocol shrinks the length of the pipe because when a node transmits a packet, only the nodes that are 3 hops away can transmit too, assuming that each hop is just within the transmission range of its one-hop neighbors but out of the range of two-hop distant neighbors (i.e., the case of the best spatial channel use). Thus, at most one packet is transmitted every 4 hops, without a possible MAC collision (if more packet transmissions occurred, MAC collisions along the path may impair packet forwarding capability and degrade throughput). Thus, the maximum pipe capacity $W_{\max}$ is defined as:

```
$W_{\max} = \text{int}(n/4)$; // round robin, $n$ is hop length
If $(n\% \, 4 \, != 0)$, $++W_{\max}$;
```

Therefore, the network pipe capacity changes according to the employed MAC protocol, with the *attenuation factor*, $W_{\max}/n$, characterized by the 802.11, fluctuating around 0.3. The receiver advertises $adwin$ according to (4), and $W_{practical}$ (*the congestion window delimiter* or, for short, *delimiter*) will yield slightly smaller $adwin$ than the actual optimum $adwin$, thereby resulting in slightly reduced throughput. As a result, the receiver's advertised window, used to signal the maximum network pipe capacity for restricting the sender's congestion window, should be:

$$W_{practical} = \frac{S_{average}}{P} \times FLD_{\min} \times \frac{W_{\max}}{n}, \tag{4}$$

where $W_{LOWER} \leq W_{practical} \leq W_{UPPER}$ (e.g., $2 \leq W_{practical} \leq W_{\max}$).

The term W_{LOWER} means the smallest size of the window, at which the sender is under heavy medium contention, and alleviates present medium contention to an extent. Loss recovery usually relies upon timeouts if any loss occurs. Above all, W_{LOWER} guarantees a minimum amount of transfer not being ceased by other heavily contending traffic loads but still sustaining for subsequent information deliveries from the sender to the receiver. The boundary values of the explicit window represent challenging situations, because the *delimiter* might be advertised close to zero when heavy contention or buffer congestion occurs (thus, consequently, other traffic can block the *rater*-responding sender's transmission[4]). For achieving global fairness among the identical TCPs operating with a *rate*, the *delimiter* can assume any value between $W_{LOWER} = 2$ (i.e., to be the same as the minimum $ssthresh$) and $W_{\max} = W_{UPPER}$, in order to mitigate heavy contention and buffer related delays—benefiting UDP traffic sources (such as VoIP or video-on-demand, which are congestion-insensitive, but with delay-sensitive high priority). This might reduce TCP flow rate because the *rater* is highly congestion-sensitive and does not make further increments unless it is allowed by the perceived FLD.

Also, a certain level of *buffer occupancy* can be used to activate the receiver's *delimiter* dependent on the buffer utilization. The use of the *buffer occupancy* metrics is useful and necessary once the receiver becomes connected with a wireline sender since it allows deciding when the *delimiter* can be invoked for buffer congestion avoidance. Similarly to the transmission window controller of TCP Vegas [27], if the perceived *buffer occupancy* exceeds a threshold predefined, it invokes the *delimiter* to

[4]In contrast, the normal TCP SACK tries to increment the window as much as possible unless a packet loss occurs, because the normal reactive TCP keeps incrementing the congestion window either exponentially or linearly until it experiences packet losses.

lower the congested buffer level, and then disables it when the *buffer occupancy* goes below another threshold, which is similar to the two thresholds of TCP Vegas. That is, the reason why we use this metrics when interconnecting with a wireline sender is because the path along the wireline link has got a built-in queuing level which under normal circumstances is tolerable. Communications within ad hoc networks do not allow buffer queuing, because of the inherent MAC related delay. Practically, the *buffer occupancy*, for which jitter is reduced by the *rater* using the smoothed forward link delay (SFLD), is calculated as:

$$\begin{aligned} Buffer\_occupancy_i &= \left(\frac{n}{FLD_{\min}} - \frac{n}{SFLD_i}\right) \\ &\times FLD_{\min} \times \frac{W_{\max}}{n}, \end{aligned}$$

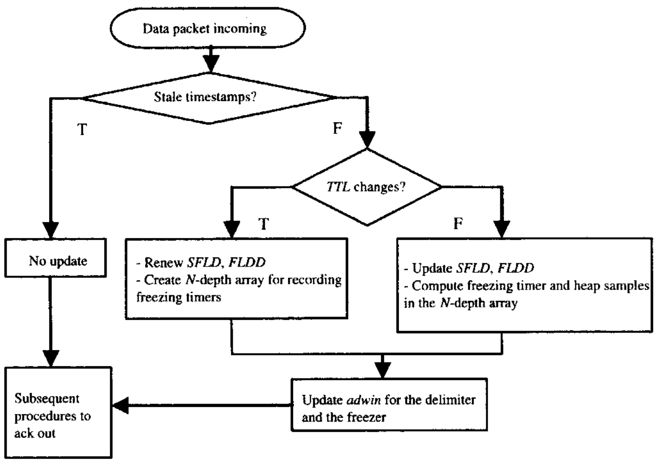
where n is the current hop length, $FLD_{\min}$ assumes the value renewed each time the value of n changes, and $SFLD_i$ is updated at every fresh timestamp of a received data packet.

The smallest and the largest value of the *buffer occupancy* for a specific hop length will be treated as the *reference values*, in order to dimension the two thresholds appropriately for identifying path anomalies; unlike TCP Vegas that has two fixed thresholds during the whole connection time, regardless of path length changes.

D. Freezing Timer

This section introduces the freezing timer that complements the *delimiter* in the case of a large discrepancy between $S_{average}$ and $S_{bottleneck}$. Against the sender's sub-optimal RTO computation, the receiver will give a feedback to freeze the sender whenever a premature RTO expiration is expected. As long as a typical link-level ACK scheme is employed, local retransmissions reduce the number of end-to-end retransmissions. However, the local retransmissions might cause unnecessary timeout(s) due to inflation of RTT and, then, invoke unnecessary end-to-end retransmission(s) with a drastic reduction of window size (reset to the initial window). In order to avoid this problem, the receiver will initiate a timer which determines when to freeze to prevent premature retransmission timeouts. At the sender, a subsequent arrival of an ACK if with an updated window will unfreeze the sender and resume the transmission.

The last instant the sender can recognize extremely high medium contention is when the sender reluctantly times out. In order to proactively prevent packet losses from medium contention-induced drops, when the receiver observes a substantial medium contention, it should propagate a freezing feedback to avoid further transmission, thereby reducing the contention level, avoiding further packet drops, and thus saving bandwidth. Ludwig [28] addressed that, (in the sense that the *retransmission timer* is roughly offset by one RTT before triggering RTO, named *retransmission timer offset*) the difference between the retransmission timer and RTO is important to determine in order to eliminate spurious timeouts because one additional RTT retained is able to avoid spurious timeouts from a sudden delay of a number of link level retransmissions as long as the RTT does not grow faster than the retransmission timer can adapt. Hence, in terms of the receiver's freezing timer and the sender's RTO,



Data packet incoming
Stale timestamps?
T
F
TTL changes?
T
F
No update
- Renew SFLD, FLDD
- Create N-depth array for recording freezing timers
- Update SFLD, FLDD
- Compute freezing timer and heap samples in the N-depth array
Update adwin for the delimiter and the freezer
Subsequent procedures to ack out

Fig. 4. Summary of the ad hoc receiver's ns implementation, where the FLD information can be updated by each packet recently received, regardless of packet sequence numbers.

the *retransmission timer offset* is beneficially providing a spare time for timely delivery of the freezing feedback, by sending a ZWA prior to timeout occurrence.

FLD fluctuation degree, (forward link delay deviation or simply $FLDD$), is similar to the RTT variance estimator ($RTTVAR$), and is calculated using a smoothed mean deviation estimator. Its applicability in ad hoc networks is based upon the assumption that the deviation of MAC related delay is an estimate of the degree of induced medium contention, since the transmission and propagation delay per packet is fixed and its most dominating factor is the level of medium contention in the 802.11. Thus, the FLDD can signify the degree of MAC contention, and is computed as follows:

$$FLDD_{i+1} = \frac{1}{4}\left|FLD_{i+1} - SFLD_i\right| + \frac{3}{4}FLDD_i, \qquad (5)$$

where $SFLD_{i+1} = \frac{7}{8}SFLD_i + \frac{1}{8}FLD_{i+1}$, $SFLD_0 = FLD_0$, $FLDD_0 = 0.5FLD_0$, $FDLL_1 = FLD_1 - SFLD_0$, and if the hop length (n) changes, then i is reset to 0, see Fig. 4.

Then we propose the freezing timer setting, similar to RTO, to validate the FLD of each received packet at the receiver. The receiver checks FLDs against the maximal allowed forward link delay time as follows:

$$\begin{aligned} Freezing\_timer(t)_i &= SFLD(t - FLD_i) \\ &+\beta \times FLDD(t - FLD_i) \end{aligned}$$

$$\text{If}(FLD_i > Freezing\_timer_i) \text{ then ZWA}.$$

In our simulations, we used $\beta = 5$, but other values are also possible.

The receiver prepares a timer by checking previous FLD samples. It determines the upper bound for the next allowable FLD. For instance, a freezing timer at the receiver is initiated immediately after the sender transmits a data packet as shown in Fig. 5. Then, if the packet arrives at the receiver, the receiver compares the measured FLD with the freezing timer in order to inspect whether the freezing timer exceeded the FLD. If the freezing timer has expired before data packet arrives (i.e., it has been exceeded), the receiver then invokes a ZWA that cancels

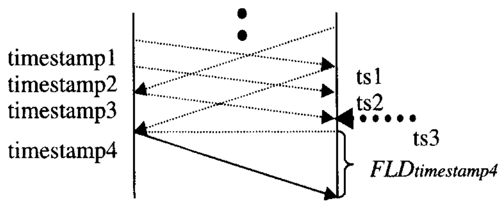


Fig. 5. Initiation of a freezing timer at the receiver.

the sender's timer and puts the sender into the persistent state until the subsequent ACK of a window update is received.

The receiver sets a freezing timer for every FLD sample obtained except when the timing information is deemed to be stale. For example, ts3, at which the last FLD sample was obtained to set $Freezing\_timer_{timestamp4}$, should be equal, or close, to timestamp4, but if they are too different, no freezing timer is set as its comparison with $FLD_{timestamp4}$ would make no sense (too stale to compare with $FLD_{timestamp4}$).

Each valid freezing timer sets an upper limit on the FLD of the next incoming data packet, and thus validates the inflation of the FLD. Freezing feedback by a ZWA which cancels the sender's timer prevents premature RTO expirations.

E. Ad Hoc Sender Enhancements

For mitigating the medium contention related congestion problems, so far the receiver has been modified to restrict the sender's congestion window, or even to freeze it, so as to alleviate the number of packet drops. This mitigates the degree of medium contention problem to the extent that data and ACK packets can keep flowing and the successive ACKs are conveyed in timely fashion to the sender. However, the sender might occasionally suffer from a packet absence recognized by duplicate ACKs, or from a number of timeouts due to sudden route breakages or network partitioning. Consequently, the sender will quench the size of the congestion window to a small value. Since the sender's activity is congestion window-limited, it is essential for the sender to recover quickly from this situation, increasing its congestion window for reflecting the available bandwidth as soon as possible.

Fig. 6 shows the transition diagram of the modified *ad hoc TCP* sender. Each state is equivalent to a state in normal TCP, except the frozen state and the operation, 'set $ssthresh$ to $adwin$'. When communicating with an ad hoc receiver, the sender is placed in the frozen state if (i) an explicit ZWA is received from the receiver, (ii) the sender has been *provisionally frozen*[5], or (iii) timeout happens. All these events cancel the retransmission timer and give a chance to re-establish $ssthresh$ to a value better adapted to the current path condition.

Immediately after the freezing, a probe timer is initiated expecting further subsequent in-flight packet(s) to be received (if any, because the sender's current retransmission timer is inadequate for frequently switching ad hoc links). On the reception of a subsequent ACK (see the procedure depicted in Fig. 7), the sender will be put into a proper state according to the received $adwin$, given that the probe timer has not expired. Otherwise, if it has already expired and probing has begun, the sender resets the retransmission timer, discarding all outstanding in-flights (without affecting $SRTT$ and $RTTVAR$, because they may inflate RTO spuriously and the sender should take subsequent ACKs to compute RTT which would be more favorable to the route currently available). Later, it goes into the slow start phase, according to the $adwin$ of the next successful probe echoed (for example, a default $adwin$, i.e., $W_{\max}$, as assumed in this study, because of packet size discrepancy[6] between probe and data packets). In the case when no in-flight packets exist, the probing is activated immediately but it maintains the current frozen regime until the first probe fails.

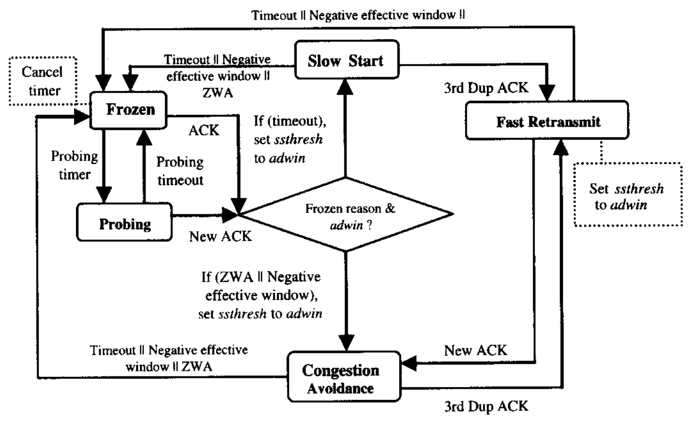

Timeout || Negative effective window ||
Cancel timer
Timeout || Negative effective window || ZWA
Slow Start
3rd Dup ACK
Frozen
ACK
If (timeout), set ssthresh to adwin
Fast Retransmit
Set ssthresh to adwin
Probing timer
Probing timeout
Probing
New ACK
Frozen reason & adwin ?
If (ZWA || Negative effective window), set ssthresh to adwin
Timeout || Negative effective window || ZWA
Congestion Avoidance
New ACK
3rd Dup ACK

Fig. 6. Transition diagram of the modified *ad hoc TCP* sender.

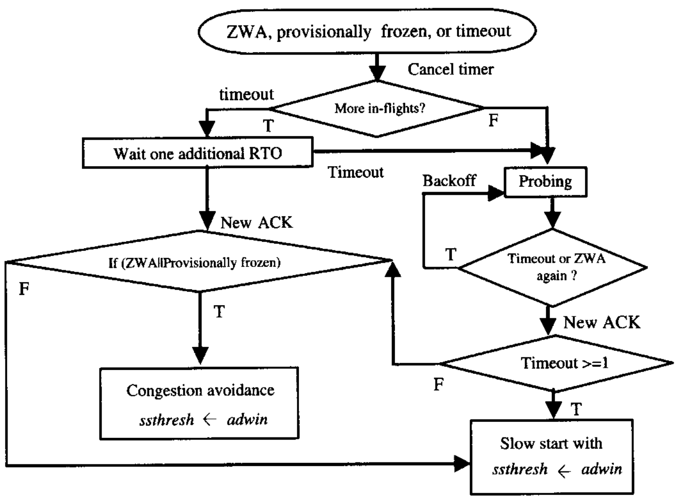

ZWA, provisionally frozen, or timeout
Cancel timer
timeout
More in-flights?
T
F
Wait one additional RTO
Timeout
Backoff
Probing
New ACK
If (ZWA||Provisionally frozen)
F
T
Timeout or ZWA again ?
T
New ACK
Congestion avoidance ssthresh ← adwin
Timeout >=1
F
T
Slow start with ssthresh ← adwin

Fig. 7. Flowchart of operations at the ad hoc sender on receiving a TCP ACK, with "one additional RTO" being used in order to prevent premature RTOs, when the sender's RTO is not optimized for dynamic ad hoc links.

[5]A negative effective window caused by a reduction of $adwin$, at which the sender will not transmit further packets, putting itself into the frozen state until a next packet arrives to reopen the window.

[6]For faster round trip time and saving bandwidth, we use a probe without TCP data payload, so it gives relatively a small FLD along the path. The receiver should adjust collected $adwin$ according to the actual data packet size, but due to the likelihood of having a new path, the receiver no longer uses current historical $SFLD$ information in responding for the next $adwin$, as well as the *freezer*. For this study, its FLD information will not be sampled for $SFLD$ and $FLDD$. Rather, the $adwin$ in reception of probes will be set to $W_{\max}$ unless otherwise adjusted and acts as a new $ssthresh$. The inadequacy of such an $adwin$ can be retrieved by some other times (e.g., ZWA, provisionally frozen, and timeout) to renew the $ssthresh$.

The probing interval, backed off if probing fails, should be adaptable to the currently available route. Holland and Vaidya [14] argued that the probing interval (a fixed interval of 2, 4, 6, 15, or 30 sec) influences the throughput of TCP, because too long an interval may prolong the discovery of new routes and, contrarily, rapid injection of probes into the network is likely to waste considerable network resources. So, they envisage an adaptive probing interval as a function of RTT or the number of hops, as it can be a more sensible way. However, it can be also flawed if round-trip path switches to a different one during the probing.

During the periods of heavy medium contention, the longer backoff timer intervals, the longer RTO can be obtained if the probe successfully round-trips. It means that the probe-induced inflated RTO can be problematic because, if the following packet is lost, it results in a prolonged loss recovery time due to the longer RTO. The probing timer should be thus limited by an upper bound. For example, the smallest throughput ($S_{\min}$) can be used here because it would be related with the current hop length and could determine an allowable maximum RTO—a maximum probing timer can be restricted to a maximum RTO ever observed since the hop length changed (or to twice the RTO for reducing probing frequency, or even much larger because of likelihood that the hop length changes to a longer one, with RTT basically exceeding the maximum). Throughput efficiency is of further concern because more frequent probings require substantial use of network resources. On the other hand, it will be unnecessary in the case of a long-term network partitioning.

In practice, if a probe does not successfully return within a certain timer, the probe timer back-offs exponentially, for up to 32 sec. If the maximum backoff interval is set to a small value, e.g., to 2 sec as in [15], in the case of stationary lengthy chains (greater than, say, 20 hops), the probe could hardly round-trip such routes within the small time period. However, in the case of dynamic topologies with the longest hop length, say, 10, this backoff interval could be set to a smaller value. The sender should be therefore aware of the hop length to the receiver in order to coordinate maximum probing backoff intervals more properly. We put this aspect aside for further study on finding the optimum probing interval.

IV. PERFORMANCE ANALYSIS

The model of the proposed scheme has been implemented in the network simulator known as ns-2.1b9a [29]. The changes were added on top of the existing TCP SACK model (i.e., sack1). In simulations, it was assumed that the participating nodes employed the AODV routing protocol, which thus imposed a certain amount of routing expense in every instant of packet forwarding. Additionally, the use of the 802.11 MAC protocol was assumed.

Performance of the scheme was assessed within a fixed time interval of 300 sec, unless otherwise stated. Simulation output data were analyzed sequentially by conducting independent replications of simulated scenarios until the relative errors of final results became smaller than 5%, at the assumed 0.95 confidence level.

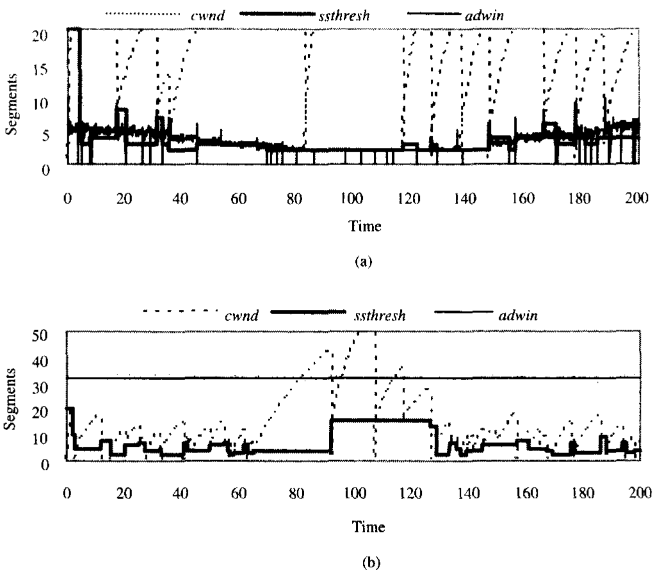


Fig. 8. Sender's behavior under (a) the *ad hoc TCP* versus (b) TCP SACK: (a) Controlled $adwin$ by the *delimiter* and ZWAs in the case of the *freezer* complying with the ad hoc sender enhancements, occasionally updating $ssthresh$ and thereby reducing the number of timeouts and fast retransmits, (b) the sender is almost always limited by the congestion window size, and has more timeouts and fast retransmits than in the case (a); a non-sensible $ssthresh$ is set by loss-oriented window adjustments.

A. Hop Length Changes

Fig. 8 illustrates the main differences between TCP SACK and the proposed scheme. The results were obtained from a single simulation run assuming the following scenario: A sender initiates an FTP transfer of packet of size 500 bytes for 200 sec to a receiver who is separated by 10 hops; each hop is 200 m long (resulting in the maximum hop length variation of at most 2). After 5 sec, the sender moves towards the receiver at 20 m/s through sites of stationary intermediate routers. After 90 sec, it stops on 10-th intermediate router for 5 sec, and then it moves back again towards the initial position at 20 m/s, arriving there after 190 sec of absence.

The results demonstrate how the proposed schemes perform for a given movement pattern, as well as how they affect the routing performance. Despite the identical node movement pattern, employing the proposed TCP scheme caused different path connectivity changes. In particular, fewer route switches occurred, so in turn smaller lower layer overheads were required during the connection time.

The simulation results resented in Fig. 9 show that the *ad hoc TCP* scheme (i.e., with *delimiter* + *freezer* + *ad hoc sender enhancements*) had the best goodput and throughput.

B. Dynamic Node Movement Patterns

In this case, 50 additional mobile nodes were added to avoid the possibility of a destination-unreachable situation due to node mobility. These nodes moved continuously within 1500 m $\times$ 300 m rectangle with the same assumed speed. The communicating mobile pairs were positioned at the edges of the network topology and remained static, so that they could share the band-

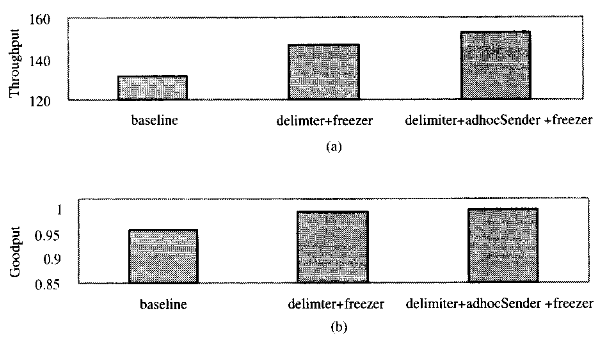


Fig. 9. Throughput (kbps) and goodput for different TCP schemes: (a) Throughput gain among three different schemes, (b) goodputs among three different schemes.

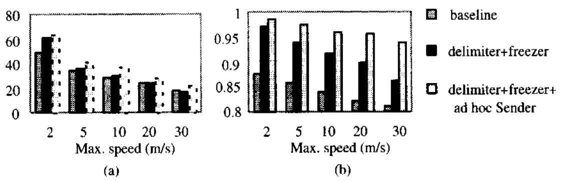


Fig. 10. A single TCP flow under different node mobilities (m/s). Maximum 15% goodput gain: (a) Throughput, (b) goodput.

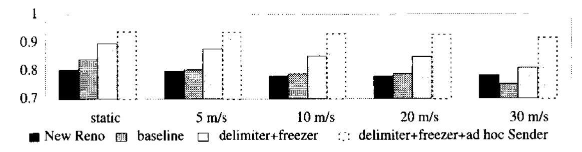


Fig. 11. Average goodputs of four TCP flows for different node mobilities. Maximum 22% goodput gain at 30 m/s.

width with each other across the same bottleneck area formed between these end nodes.

As shown in Figs. 10 and 11, our *ad hoc TCP* outperformed the others in terms of both throughput and goodput. Yet, we can see that at high node mobility, the "receiver-only enhancements" case, i.e., if *delimiter* and *freezer* are used, throughput is lower than in the case of the baseline TCP SACK. This was the result of relatively fewer in-flight packets under our scheme that under the baseline protocol and no guarantee of timely deliveries of ACKs, and ZWAs if advertised, causing degradation of throughput due to RTO backoffs. However, one can see a consistently better goodput gain under the "receiver-only enhancements" case also in these cases.

When comparing Figs. 12(a) and (b), with Fig. 12(c), one can see that a fair share of the medium can be achieved also in a network with the *ad hoc TCP* and New Reno flows, except that when node mobility is high, since then New Reno flows slightly steal bandwidth from the *ad hoc TCP* ones. New Reno, as well as the baseline, being loss-based is likely to cause large bursts of data, so increasing buffer utilization and causing RTT to increase. Subsequent RTO inflation results in relatively longer timeouts. Yet, this trend of large RTOs helps with attaining friendliness with the *ad hoc TCPs*, as evident in Fig. 12. However, let us not that New Reno will suffer because of large RTOs. Thus, the *ad hoc TCP*, whose transmission window $W_{practical}$ (see Section III-C) is eventually reduced to 2 due to the aggres-

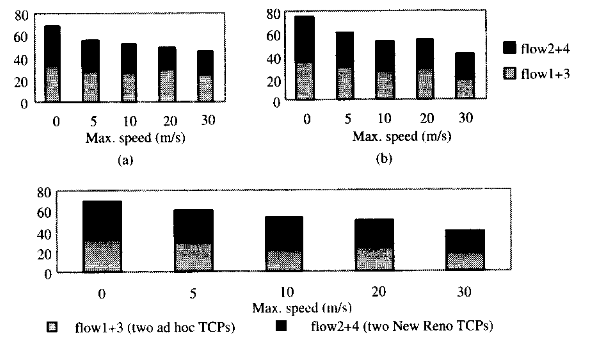


Fig. 12. Throughputs of the four TCP flows for different node mobilities: (a) Four New Reno, (b) four *ad hoc TCPs*, (c) two *ad hoc TCPs* and two New Reno TCPs.

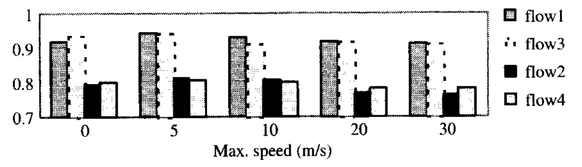


Fig. 13. Goodputs of disparate TCP flows. Flows 1 and 3 are *ad hoc TCPs*, 2 and 4 are New Reno TCPs.

sive New Reno, can transmit. As a result, to an acceptable degree, the *ad hoc TCP* can co-exist with the greedy reactive TCP in terms of throughput. However, Fig. 13 shows that the goodput of the *ad hoc TCP* is superior to the one of New Reno TCP by up to 19.4%. This clearly shows the advantage of the network-wide deployment of the *ad hoc TCP*.

V. CONCLUSIONS

In this paper, we evaluated the performance of the baseline TCP SACK and confirmed that it can both over-utilize and under-utilize the available bandwidth. We then demonstrated that the combination of our newly proposed schemes, i.e., *delimiter* plus *freezer*, with or without *ad hoc sender enhancements*, outperforms, and can co-exist in a friendly manner with, the baseline TCP SACK, in the sense of both throughput and goodput.

In brief, the primary features of our *ad hoc TCP* are listed below:

- This is purely end-to-end scheme that does not require any network-originated feedback.
- It is characterized by high packet delivery ratio and is power and bandwidth effective.
- It reduces unnecessary RTO expirations-by means of the *delimiter* and the *freezer*.
- It is characterized by a fast adaptability to hop length changes—each time the hop length changes, the receiver renews the measurements (e.g., $SFLD$ and $FLDD$).
- No exponential RTO backoffs are needed because they are replaced with probing backoffs.
- It offers a global fairness—achieved by applying identical *raters*.

Further studies are needed on calibration of the coefficients of the *freezer* to optimize its performance and on revision of the sender's historical measurements of $SRTT$, $RTTVAR$, and RTO for using them in ad hoc networks. Also, dynamic sender's congestion window progression strategy is required. One could also consider intelligent strategies for choosing the probing frequency in order to conserve the scarce network resources. Extending the prototype of the receiver-oriented flow control (i.e., *delimiter* and *freezer*) to *ad hoc-cum-wired* networks, with base stations that supervise ad hoc associated links, could be also desirable.

REFERENCES

[1] M. S. Corson and J. Macker, "Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations," *RFC 2501*, Internet Engineering Task Force, Jan. 1999.

[2] K. Sundaresan *et al.*, "ATP: A reliable transport protocol for ad hoc networks," in *Proc. MobiHoc 2003*, June 2003.

[3] S. Mascolo *et al.*, "TCP Westwood: End-to-end bandwidth estimation for efficient transport over wired and wireless networks," in *Proc. ACM Mobicom 2001*, 2001.

[4] S. Floyd *et al.*, "An extension for the selective acknowledgement (SACK) option for TCP," *RFC 2883*.

[5] R. Ludwig and R. Katz, "The Eifel algorithm: Making TCP robust against spurious retransmissions," *ACM Computer Commun. Review*, Jan. 2000.

[6] C. Parsa and J. J. Garcia-Luna-Aceves, "TULIP: A link-level protocol for improving TCP over wireless links," in *Proc. IEEE WCNC'99*, 1999, pp. 1253–1257.

[7] Z. Fu *et al.*, "On TCP performance in multihop wireless networks," University of California at Los Angeles, 2003.

[8] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. 15-th Int. Conf. Dist. Computing Syst.*, May 1995, pp. 136–143.

[9] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM Computer Commun. Review*, vol. 27, pp. 19–43, Oct. 1997.

[10] H. Balakrisnan, S. Seshan, and R. H.Katz, "Improving reliable transport and handoff performance in cellular wireless networks," in *Proc. Mobicom'95*, Nov. 14–15, Berkeley, (California, USA), 1995.

[11] K. Ratnam and I. Matta, "WTCP: An efficient transmission control protocol for networks with wireless links," in *Proc. IEEE Symp. Computer and Commun. (ISCC)*, June 1998.

[12] D. Sun and H. Man, "Evaluation of reliable data transport schemes for mobile ad hoc networks," 2002.

[13] T. Goff *et al.*, "Freeze-TCP: A true end to end TCP enhance mechanism for mobile environments," in *Proc. IEEE INFOCOM 2000*, 2000.

[14] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proc. IEEE/ACM Mobicom'99*, Aug. 1999.

[15] J. P. Monks, P. Sinha, and V. Bharghavan, "Limitations of TCP-ELFN for ad hoc networks," in *Proc. 7-th Int. Workshop on Mobile Multimedia Commun. (MoMuC)*, 2000.

[16] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," *IEEE J. Select. Areas Commun.*, vol. 19, no. 7, pp. 1300–1315, July 2001.

[17] K. Chandran *et al.*, "A feedback based scheme for improving TCP performance in ad hoc wireless networks," in *Proc. 18-th Int. Conf. Dist. Computing Syst.*, May 1998.

[18] D. Kim, C.-K. Toh, and Y. Choi, "TCP-BuS: Improving TCP performance in wireless ad hoc networks," in *Proc. IEEE ICC 2000*, New Orleans, LA, June 2000.

[19] K. Chen and K. Nahrstedt, "EXACT: A explicit rate-based flow control framework in MANET (extended version)," University of Illinois at Urbana-Champaign, July 2002.

[20] F. Wang and Y. Zhang, "Improving TCP performance over mobile ad hoc networks with out-of-order detection and response," in *Proc. ACM Mobihoc 2002*, June 2002.

[21] T. D. Dyer and R. V. Boppana, "A comparison of TCP performance over three routing protocols for mobile ad hoc netowrks," in *Proc. ACM Mobihoc 2001*, Oct. 2001.

[22] Z. Fu *et al.*, "Design and implementation of a TCP-friendly transport protocol for ad hoc wireless network," in *Proc. 10-th IEEE Int. Conf. Network Protocols*, Paris, France, 2002, pp. 212–225.

[23] R. Oliveira, T. Braun, and M. Heissenbuttel, "An edge-based approach for improving TCP in wireless mobile ad hoc networks," Berne University, Switzerland, 2002.

[24] G. S. Ahn *et al.*, "SWAN: Service differentiation in stateless wireless ad hoc networks," in *Proc. IEEE INFOCOM 2002*, New York, June 2002.

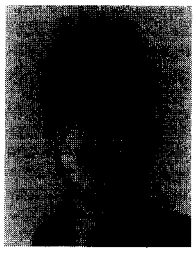
[25] S. Xu and T Saadawi, "Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc network?" *IEEE Commun. Mag.*, vol 39, no. 6, pp. 130–137, June 2001.

[26] X. Hannan, "A flexible quality of service model for mobile ad hoc networks," Ph.D. dissertation, Singapore University, Singapore, 2002.

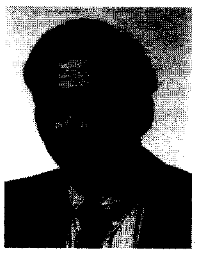
[27] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. SIGCOMM'94*, 1994.

[28] R. Ludwig, "A case for flow–adaptive wireless links," Technical Report, University of California, Berkeley, 1999.

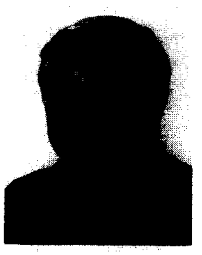
[29] K. Fall and K. Varadhan, "NS notes and documentation," Technical Report, The VINT Project, Dec. 2003.



Sung Rae Cho was born in Busan, Korea, on 17 November 1975. He received a B.E. and M.E. degrees (both from Electronic eng.) from the University of Canterbury in Christchurch, New Zealand, in 2001 and 2004, respectively. Presently, he is working in the Basic Research Laboratory, Communication Device Module Team, at ETRI, Korea. His research interests include low power-aware system design, security, and ad hoc networks.



Harsha Sirisena has a B.Sc. (Eng) (Honors) degree from the University of Ceylon, Sri Lanka, and a Ph.D. degree from the University of Cambridge, UK. He is currently a Reader in Electrical and Computer Engineering at the University of Canterbury, in Christchurch, New Zealand. The author of over 120 research papers; has held visiting positions at Lund Institute of Technology, University of Minnesota, Australian National University, and National University of Singapore. His research interests are in wireless and optical networks, transport and application layer congestion control, and soft computing. He is Senior Member of IEEE and member of IEICE (Japan).



Krzysztof Pawlikowski has an M.Sc. (Computer Engineering) and Ph.D. (with Dinstinction) degrees from the Technical University of Gdansk, Poland. He is a Professor in Computer Science and Software Engineering at the University of Canterbury, in Christchurch, New Zealand. The author of over 120 research papers and four books; has given invited lectures at over 80 universities and research institutes in Asia, Australia, Europe, and North America. He was an Alexander-von-Humboldt Research Fellow (Germany) in 1983–84 and 1999. His research interests include performance modelling of telecommunication networks, discrete-event simulation, and distributed processing. He is Senior Member of IEEE and member of ACM and ISCS (Int. Soc. for Computer Simulation).