A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks[*]

Thomas D. Dyer
Computer Science Division
The Univ. of Texas at San Antonio
San Antonio, TX 78249
tdyer@cs.utsa.edu

Rajendra V. Boppana
Computer Science Division
The Univ. of Texas at San Antonio
San Antonio, TX 78249
boppana@cs.utsa.edu

ABSTRACT

We examine the performance of the TCP protocol for bulk-data transfers in mobile ad hoc networks (MANETs). We vary the number of TCP connections and compare the performances of three recently proposed on-demand (AODV and DSR) and adaptive proactive (ADV) routing algorithms. It has been shown in the literature that the congestion control mechanism of TCP reacts adversely to packet losses due to temporarily broken routes in wireless networks. So, we propose a simple heuristic, called fixed RTO, to distinguish between route loss and network congestion and thereby improve the performance of the routing algorithms. Using the *ns-2* simulator, we evaluate the performances of the three routing algorithms with the standard TCP Reno protocol and Reno with fixed RTO. Our results indicate that the proactive ADV algorithm performs well under a variety of conditions and that the fixed RTO technique improves the performances of the two on-demand algorithms significantly.

1. INTRODUCTION

The TCP protocol has been extensively tuned to give good performance at the transport layer in the traditional wired network environment. However, TCP in its present form is not well-suited for mobile ad hoc networks (MANETs) where packet loss due to broken routes can result in the counter-productive invocation of TCP's congestion control mechanisms. Although a number of studies have been conducted and protocol modifications suggested, improving TCP performance in MANETs is still an active area of research. The performance studies published to date have either focussed on specific protocols or were restricted to a single TCP connection.

The objectives of this study were two-fold: to compare TCP performance over different proposed MANET routing protocols, and to explore the utility of using a sender-based heuristic to distinguish between packet loss due to congestion and loss due to route failures. We conducted simulations to measure the performance of TCP for bulk-data transfers over two of the proposed on-demand protocols, DSR [16] and AODV [18]. A third protocol, ADV [5], was also considered. ADV combines an on-demand approach with proactive distance vector routing. A background UDP traffic load from constant bit rate sources was included so that we could assess the impact of non-TCP traffic on the TCP throughput and vice versa.

We examined the relative impact of two existing options designed to enhance TCP performance, selective acknowledgements and delayed acknowledgements. We also considered a modification to the TCP sender designed to lessen the negative effect of TCP's reaction to retransmit timeouts caused by temporary route failures. Several researchers have designed mechanisms by which the TCP sender is notified of a route failure, and in some schemes, the sender is also notified when the route has been re-established [7, 13]. We have not implemented such a scheme for this study, but our sender-side heuristic gives some indication of the performance gains that can be achieved by altering the TCP sender's behavior.

The results of our simulations demonstrate that the proactive ADV protocol performs well under a variety of conditions. ADV does not, however, benefit from our modifications to the TCP sender. On the other hand, the on-demand algorithms both showed significant performance improvements with the addition of our sender-side heuristic to TCP Reno.

2. RELATED WORK

Several performance evaluations of MANET routing protocols for UDP traffic have been presented in the literature [5, 6, 9, 15, 17]. Other studies have presented mechanisms for improving TCP performance in 1-hop wireless networks [3, 4, 2, 20].

Recent studies have addressed the TCP performance problems caused by route failures in an ad hoc network. Chandran et al. proposed a feedback-based scheme called TCP-Feedback or TCP-F [7]. In this scheme, when an intermediate node detects the disruption of a route due to the mobility of the next host along that route, it explicitly sends a Route Failure Notification (RFN) to the TCP sender. Upon receiving

[*]This research has been partially supported by DOD/AFOSR grant F49620-96-1-0472 and NSF grant CDA 9633299.

ACM Symposium on Mobile Ad Hoc Networking & Computing (Mobihoc), October 2001

the RFN, the source suspends all packet transmissions and freezes its state, including the retransmission time out interval and the congestion window. When an intermediate node learns of a new route to the destination, it sends a Route Reestablishment Notification (RRN) to the source, which then restores its previous state and resumes transmission. The effect of this scheme was studied by simulating a single TCP connection. The main conclusion of the study was that average route repair time has a major impact on TCP throughput.

Holland et al. advocate the use of explicit link failure notification (ELFN) to significantly improve TCP performance in MANETs [13]. In the ELFN scheme, when the TCP sender is informed of a link failure, it freezes its state (timers and window size) as in TCP-F. There is no route re-establishment notification, however. Instead, the source sends out packets (probes) at regular intervals to determine if a new route is available. Using *ns-2* [11] to simulate a IEEE 802.11 wireless network running TCP Reno and the DSR routing protocol, they show that the ELFN scheme can increase TCP performance considerably. They also obtained significant performance improvements by simply turning off the DSR feature whereby intermediate nodes send out route updates based on the contents of their (often stale) route caches.

Ahuja et al. [1] conducted a simulation-based comparison of TCP performance over several MANET routing protocols, including AODV, DSR, and SSA [10]. Only a single source of TCP traffic was simulated. They concluded that frequency of route failures, routing overhead, and delay in route establishment are the important determinants of TCP throughput in an ad hoc network. Like [13], they found that disabling route replies from cache actually improved TCP throughput by eliminating the effect of stale routes.

3. PROTOCOLS

In this section, we give a brief overview of the routing protocols used in our performance analysis. We also discuss the variations of the TCP protocol that were considered.

Routing protocols. As noted in the introduction, AODV and DSR are *on-demand* algorithms. Unlike *proactive* protocols such as DSDV [19], on-demand protocols do not maintain routes between all the nodes in an ad hoc network. Rather, routes are established when needed through a route discovery process in which a route request is broadcast. A route reply is returned either by the destination or by an intermediate node with an available route. Route error messages are used to invalidate routing table entries when link failures are detected.

Like AODV, ADV [5] is a distance vector algorithm that uses sequence numbers to avoid long-lived routing loops [12, 19]. Routes to active receivers are established when needed as in an on-demand algorithm. Routing updates, however, are used to distribute routing information in a proactive fashion. ADV has no explicit route repair mechanism, relying instead on the routing updates to re-establish broken routes. Unlike the periodic updates in a traditional distance vector algorithm, ADV routing updates are triggered adaptively in response to network load and topology changes. In particular, a node can inform its neighbors, as part of its routing updates, of its need to replace a broken route, which in turn allows the neighboring nodes to trigger updates much more quickly. This feature makes ADV more adaptive than the previous distance vector algorithms.

Transport protocols. We used mixed TCP and UDP traffic. For UDP, we used the existing standard UDP protocol.

For TCP, we took the Reno version of the TCP protocol as our base case. We compared this baseline performance to that obtained using two available TCP options, selective ackowledgements (SACK) and delayed acknowledgments. Considering the frequency with which routes can change in a MANET, we assumed that out-of-order and delayed packet (and ACK) deliveries would occur somewhat frequently. The use of SACK should reduce the number of unnecessary retransmissions that can result when duplicate ACKs caused by out-of-order deliveries trigger fast retransmits. Delayed ACKs are expected to help by reducing the volume of ACK traffic in normal network conditions. In the case of a broken route, delayed ACKs allow the TCP sender to increase its send window in increments larger than 1. When two delayed data packets arrive at the destination in sequence, only one ACK will be generated and the sender does not retransmit any additional packets.

In the event of a retransmit timeout, TCP retransmits the oldest unacknowledged packet and doubles the retransmit timeout interval (RTO). This process is repeated until an ACK for the retransmitted packet has been received. This exponential backoff of the RTO enables TCP to handle network congestion gracefully. However, in a MANET, the loss of packets (or ACKs) may be caused by temporary route loss as well as network congestion. Since routes are likely to be broken frequently in high node mobility environments, routing algorithms for MANETs are designed to repair broken routes quickly. To take advantage of this capability, it is intuitive to let a TCP sender retransmit the unacknowledged packet at periodic intervals rather than having to wait increasingly long periods of time between retransmissions.

Therefore, we modified the TCP sender, employing a heuristic to distinguish between route failures and congestion without relying on feedback from other network nodes. When timeouts occur consecutively, i.e. the missing ACK is not received before the second RTO expires, this is taken to be evidence of a route loss. The unacknowledged packet is retransmitted again but the RTO is not doubled a second time. The RTO remains fixed until the route is re-established and the retransmitted packet is acknowledged.

Our modification to the TCP congestion control mechanism is intended for use in wireless networks only. The *getsockopt* and *setsockopt* function calls in Unix may be easily modified so that applications can make use of this feature.

4. SIMULATION METHODS

For our simulations, we used the *ns-2* network simulator [11] with the CMU extensions by Johnson et al. [8]. These extensions include the modeling of an IEEE 802.11 wireless LAN [14]. We used CMU's implementation of DSR, and all parameter values and optimizations used for DSR are as described by Broch et al. [6]. The AODV and ADV implementations are by the AODV and ADV groups, respectively. For

AODV we used the following settings: MAC link layer feedback; 50s active route timeouts; local route repair; 1, 2, and 7 for TTL_START, TTL_INCREMENT, and TTL_THRESHOLD, respectively. For ADV, all parameter values, except buffer timeout (which is set to 30s in this study), are the same as those given in [5].

The network we simulated consisted of 50 nodes randomly placed on a 1000m x 1000m field at the beginning of a simulation. We utilized a mobility pattern based on the *random waypoint* model. To mimic high node mobility, node speeds were uniformly distributed between 0 m/s and 20/ms, yielding a mean node speed of 10 m/s, and only zero-length pause times were considered.

We simulated the *steady-state* conditions of a network with various background traffic loads generated by 10 and 40 constant bit rate (CBR) connections. The CBR packet sizes were fixed at 512 bytes. After a warm-up time of 100 seconds, one or more TCP connections were established over each of which an FTP file transfer was conducted for 900 seconds. The TCP packet size was 1460 bytes, and the maximum size of both the send and receive windows was 8. Since routing protocol performance is sensitive to movement patterns, 50 different mobility patterns (scenarios) were generated.

In each simulation run, we measured connect time, throughput, and goodput. Connect time is the time it takes to deliver the first TCP packet. Short connect times are important for some types of TCP traffic such as HTTP. Throughput is computed as the amount of data transferred by TCP divided by 900 seconds, the time interval from the end of the warm-up period to the end of the simulation. This does not include redundant packet receipts due to unnecessary packet retransmissions and packet replication in the network. Goodput is the ratio of TCP packets successfully delivered to the total number of TCP packets transmitted. In order to gauge the routing protocol overhead, we measured both the number of routing packets and the number of bytes of routing data transmitted per second at the IP layer. The overhead includes the routing of the background CBR traffic. For DSR, the number of bytes of routing data transmitted includes the routing information carried by data packets. We also measured the number of routing packets transmitted per second at the MAC layer, including all the IP layer routing packets and the RTS, CTS, and ACK control exchange packets used for transmitting unicast data *and* routing packets.

5. PERFORMANCE RESULTS

For each number of TCP connections (1, 2, 5, and 10), we performed a series of four simulation runs. Each simulation run tested a different technique or combination of techniques: TCP Reno, Reno with SACK, Reno with SACK and delayed ACKs, and fixed RTO on consecutive timeouts plus SACK and delayed ACKs. In each run, a set of performance measurements were made for each of the three routing protocols at each of several background traffic loads from 10 CBR connections and from 40 CBR connections.

5.1 1 TCP Connection

Figures 1 and 3 show the connect times, throughputs, goodputs, and routing overheads, averaged over the 50 scenarios, observed for each of the protocols for 1 TCP Reno connection with a background traffic load generated by 10 and 40 CBR connections. In figures 2 and 4, TCP's SACK and delayed acknowledgment options have been added along with the fixed-RTO mechanism. While the use of SACK alone and the combination of SACK and delayed ACKs did enhance performance in some cases (10-12% increases in throughput for AODV and DSR at higher traffic loads, for example), the gains were modest and those results are not included here.

With TCP Reno and a 50 Kbps 10-CBR background, DSR throughput was about 55% that of the other protocols. This is a reasonable result considering that with a lighter background traffic load, routes in the network are more likely to be stale, and stale routes are troublesome for DSR. The stale route problem is very evident when we consider the connect times. While the connect times for ADV and AODV remain essentially unchanged with the addition of the fixed RTO, the connect time for DSR dropped dramatically from over 30 seconds to just under 6 seconds. At the same time, DSR throughput increased by 67%. The fixed-RTO technique continued to yield a 70-75% gain in DSR throughput as the 10-CBR traffic increased from 50 Kbps to 200 Kbps. Significant throughput gains were also observed for the 40-CBR case as shown in Figures 3 and 4. The goal of fixing the RTO was to reduce the impact of route unavailability. It appears this technique was particularly effective in mitigating the stale route problem for DSR.

The fixed-RTO technique, in combination with SACK and delayed ACKs, also yielded increased throughput for AODV, although the gains were much smaller than those observed for DSR. Interestingly, the increase in throughput for the 40-CBR case was about twice that seen in the 10-CBR case. As the background traffic load increased, the gain in throughput remained the same, about 8%, for 10 CBR connections, while the gain in the 40-CBR case grew to 48% for a 200 Kbps load. To the extent that the additional routing traffic from 40 CBR flows results in increased packet delays, we would expect the fixed-RTO technique to be of relatively greater benefit.

The increases in throughput that we observed for ADV did not exceed 4%. It appears ADV was performing as well as possible since, although the application of these techniques tended to minimize the performance differences among the protocols, in no case did the other protocols exhibit a higher level of throughput than ADV. Furthermore, in the 40-CBR case, ADV clearly outperformed AODV and DSR regardless of which techniques were used. We observed this same result at higher background traffic loads. For a 100 Kbps load and 40 connections, ADV throughput was 17% to 52% greater than that of the other protocols.

The fixed-RTO technique lets the TCP sender avoid waiting for ever longer periods of time before attempting to retransmit the next unacknowledged packet. In effect, the TCP sender is probing the network in a manner similar to the ELFN scheme [13]. But because the probe interval is equal to the retransmit time interval which in turn is tied to the estimated RTT, the fixed-RTO method is inherently adaptive.

The graphs in Figures 5 - 7 plot congestion window size as it changes over time for the scenario in which the three protocols yielded their worst or nearly worst performance. The

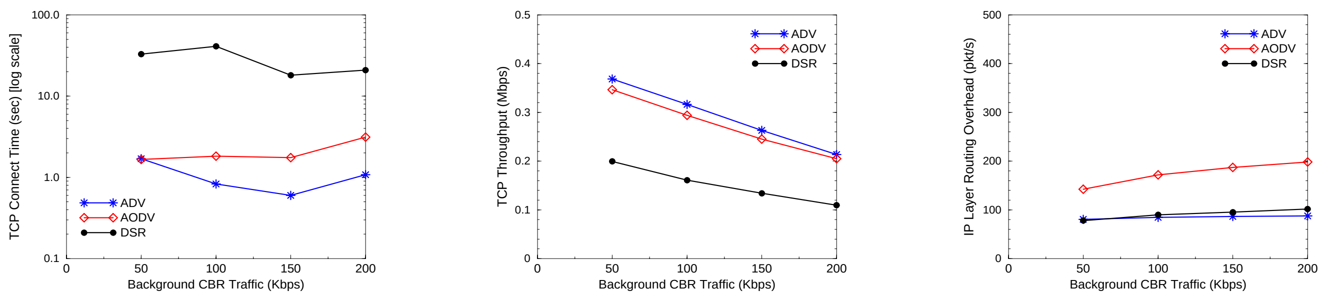

Figure 1: Connect times, throughputs, and routing overhead for 1 TCP Reno connection with a 10-CBR background.

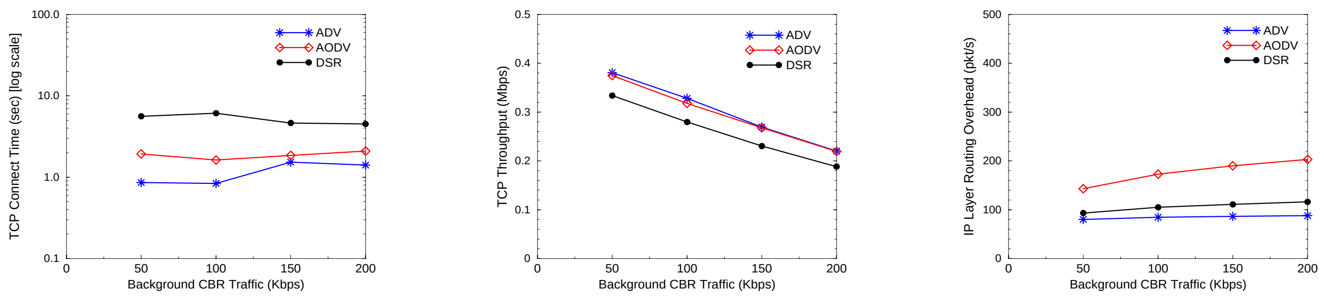

Figure 2: Connect times, throughputs, and routing overhead for 1 TCP connection using SACK + delayed ACKs + fixed RTO with a 10-CBR background.

plotted window size is a 5-second moving average, so non-integral window sizes appear. A 5-second interval was chosen to smooth the plot without losing important detail. The first 100 seconds were the warmup period before the TCP traffic was initiated.

In this scenario, the length of the shortest possible path between the TCP sender and receiver nodes changed fairly frequently and tended to be a bit long, often 5 or 6 hops or more. Around 375 seconds into the simulation, all three protocols experienced a route failure. Referring to Figures 5 - 7, we see that, with TCP Reno, ADV was able to recover fairly quickly, but AODV and DSR were stalled for extended periods of time. In Figure 6, the AODV congestion window is stuck at its minimum value of 1 from the route loss around 375 seconds until after 600 seconds, then again from about 800 seconds until the end of the run. The resulting throughputs for AODV and DSR were approximately 91 Kbps and 36 Kbps, respectively. ADV throughput was higher, but still just 170 Kbps. The route repair and route discovery mechanisms of the on-demand protocols were not able to cope with the high degree of mobility and the large number of hops from sender to receiver. ADV, with its proactive routing, was able to adapt to the rapidly changing network topology, keeping the congestion window open. With the addition of the fixed RTO, the more frequent packet retransmisions caused AODV and DSR to initiate route discoveries which led to faster route repairs, resulting in a larger congestion window on average. AODV and DSR throughputs increased to 202 Kbps and 130 Kbps, respectively. ADV, on the other hand, did not receive any benefit because the routing information disseminated through triggered updates was usually sufficient to re-establish routes before consecutive timeouts occurred. As a result, ADV throughput remained virtually unchanged.

5.2 Multiple TCP Connections

In the case of multiple TCP sources, we considered background traffic loads of 100 Kbps and 200 Kbps from 10 CBR and 40 CBR connections. The sender and receiver nodes were unique for each connection, although in some cases a TCP end point was also the endpoint of one or more CBR flows. The combined throughputs of 2, 5, and 10 TCP connections with a 100 Kbps background traffic load are shown in Figures 8 and 9.

As was observed previously for 1 TCP source, the addition of SACK and delayed ACKs to TCP Reno resulted in modest gains (5-10%) in throughput. In most cases, ADV continued to provide the highest throughput. As before, AODV showed decreased throughput relative to ADV and DSR as the number of CBR connections increased from 10 to 40. Because AODV relies on its route discovery process to establish new routes and repair broken routes, the larger number of connections results in considerably more work. At the same time, a larger number of connections, as well as a higher volume of traffic, enables DSR to use caching and snooping effectively to reduce this route discovery overhead. For 5 and 10 TCP sources, DSR throughput was observed to be nearly as high or even higher than that of AODV, particularly for a larger number of CBR flows.

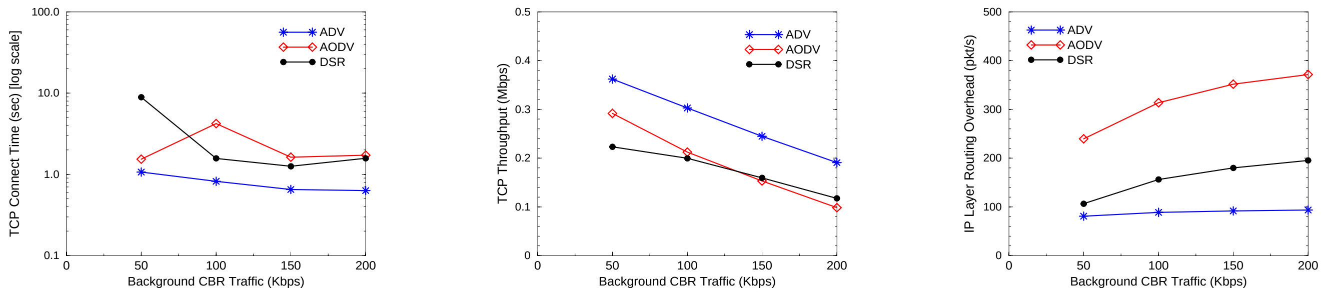

Figure 3: Connect times, throughputs, and routing overhead for 1 TCP Reno connection with a 40-CBR background.

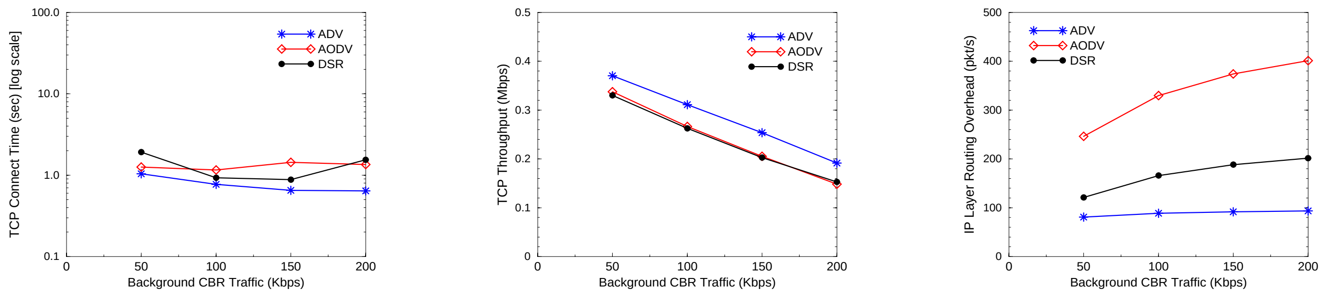

Figure 4: Connect times, throughputs, and routing overhead for 1 TCP connection using SACK + delayed ACKs + fixed RTO with a 40-CBR background.

When the fixed-RTO technique was employed, the performance differences of the three protocols tended to be minimized. However, for more than two TCP connections, the benefit of fixing the RTO in response to consecutive timeouts became great enough that AODV and DSR provided greater throughput than did ADV. For 10 TCPs with a 10-CBR background, DSR throughput was 10% higher than ADV throughput. This effect became even more pronounced when the background traffic load was increased to 200 Kbps as can be seen in Figures 10 and 11.

With a 200 Kbps traffic load from 10 CBR flows, AODV and DSR both performed better relative to ADV. As we show later in this report, ADV provided significantly better CBR throughput than the other protocols. Therefore, as the volume of CBR traffic increased, the impact of the background load on TCP throughput was largest for ADV. For the unmodified TCP sender (i.e., no fixed RTO), ADV and AODV throughputs were virtually the same and were higher than the DSR throughput, although this advantage decreased as the number of TCPs was increased. With 40 CBR connections, ADV continued to provide better throughput than both AODV and DSR, which had nearly identical performance.

The connect times, throughputs, goodputs, and routing overheads observed for the various number of TCP connections with a 100 Kbps background traffic load are shown in Figures 12 - 15. Given the relatively small throughput gains observed when using just the TCP options, we show only the results obtained using TCP Reno and TCP Reno with SACK, delayed ACKS, and fixed RTO. The relative performances of the three protocols for a 200 Kbps background traffic load were not qualitatively different from those observed for a 100 Kbps load, so to save space those results are not included.

For TCP Reno, the goodputs we observed for all three protocols were similar, ranging from about 96% to 98%. DSR generally achieved the highest goodput, and AODV the lowest. When the TCP options and the fixed-RTO mechanism were added, goodputs ranged from about 94% to 96%. ADV had the highest goodput in most cases, and again AODV had the lowest. Goodput decreased slightly as the number of TCP connections was increased.

AODV generated the highest number of routing packets, followed by DSR. The routing activity for AODV and DSR increased as more TCPs were added, but the rate of increase diminished as the number of connections grew. Due to its proactive nature, ADV generated the fewest routing packets, and ADV routing activity was constant with respect to both the number of TCP connections and the number of CBR connections.

Although the frequency of ADV routing updates remained constant, the amount of routing information contained in the updates did increase with the number of TCPs. When measured in Kbps, ADV routing overhead was quite a bit higher than that of the other protocols. For 1 TCP flow, ADV generated about twice as many routing bytes per second as AODV for 40 CBR connections, again a consequence of ADV's proactive routing. Several researchers have pointed out that in a wireless network, the high cost of accessing the medium places a premium on a reduced number of routing packets.

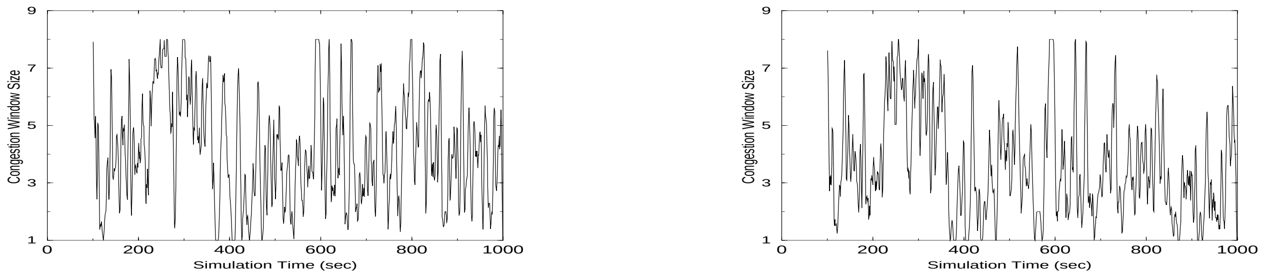

Figure 5: ADV congestion window sizes of a TCP connection using Reno and SACK + delayed ACKs + fixed RTO.

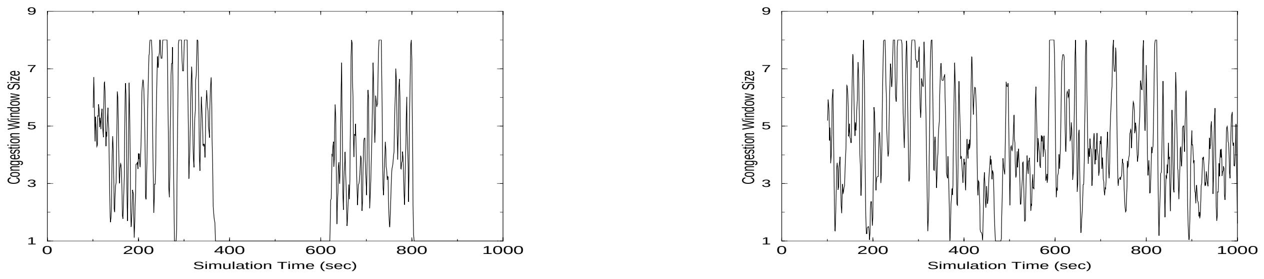

Figure 6: AODV congestion window sizes of a TCP connection using Reno and SACK + delayed ACKs + fixed RTO.

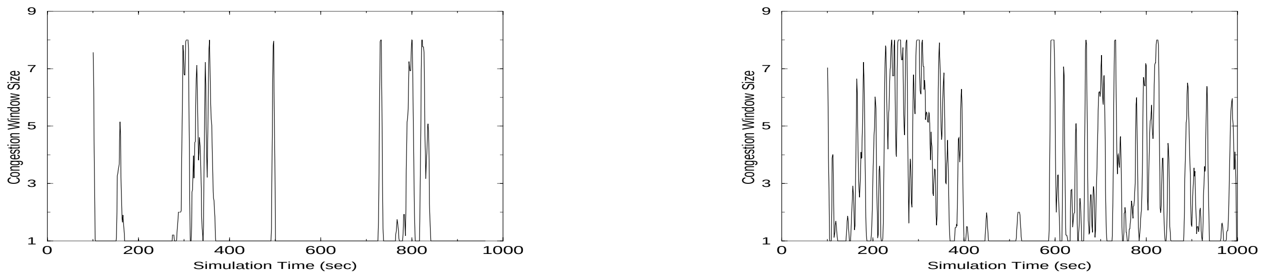

Figure 7: DSR congestion window sizes of a TCP connection using Reno and SACK + delayed ACKs + fixed RTO.

Hence, the larger number of ADV routing bytes may not be a concern.

The TCP performance afforded by a routing protocol should be weighed against how well the protocol is able to move non-TCP traffic at the same time. To assess the impact of TCP traffic on non-reactive CBR flows, we measured the average CBR packet latency and the fraction of CBR packets successfully delivered. The results for 10 CBR connections are shown in Figures 16 - 17.

CBR latencies increased as more TCPs were added, but the increases were not enough to indicate saturation. ADV latency was about twice that observed for the other protocols, which had nearly identical latencies. This is in contrast to the results reported by [5], who found ADV latencies were lower than those of AODV and DSR. In that study, however, the CBR packet size was only 64 bytes, and a very short buffer refresh time of 1 second was used for ADV. For 10 TCP connections, ADV latency was slightly greater than 1 second compared to 0.5 second for DSR and AODV. The use of a fixed RTO had essentially no impact compared to TCP Reno.

ADV did a much better job of handling the background CBR flows in terms of packet delivery fraction. With 10 TCP traffic sources, ADV delivered 70-75% of the CBR packets compared to about 60% for AODV and 50% for DSR. For only 2 TCPs, ADV achieved a delivery fraction above 90% for 10 CBR flows and in excess of 95% for 40 CBRs. AODV outperformed DSR in all cases, the observed difference increasing with the number of TCP and CBR connections. Again, the use of a fixed RTO had little or no effect compared to TCP Reno.

6. CONCLUSIONS

Using the well-known *ns-2* simulator with 802.11 wireless LAN extensions, we compared the performances of two on-demand algorithms, AODV and DSR, and a proactive algorithm, ADV. We varied the number of TCP connections, the

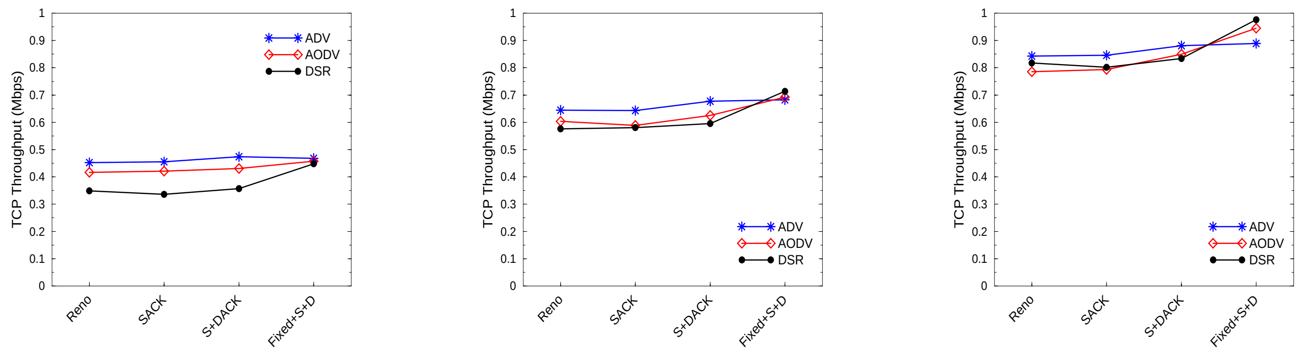

Figure 8: Combined throughputs for 2, 5, and 10 TCP connections with a 100 Kbps, 10-CBR background.

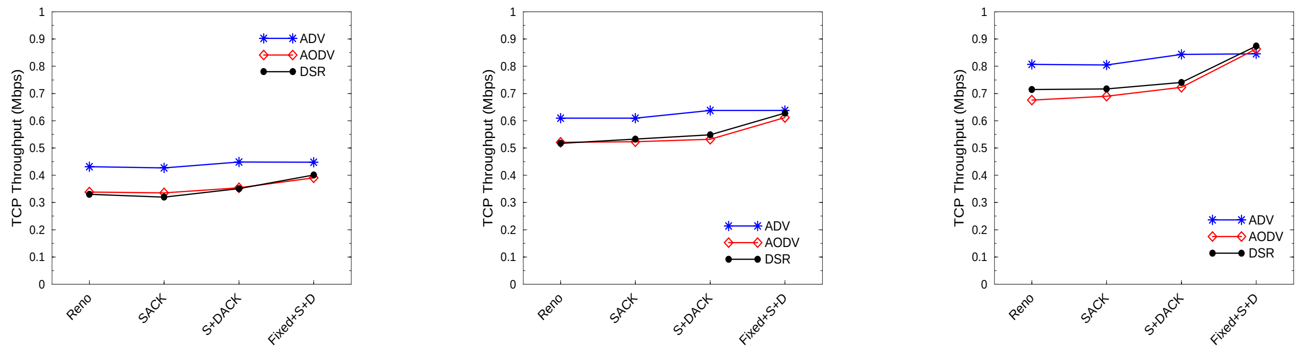

Figure 9: Combined throughputs for 2, 5, and 10 TCP connections with a 100 Kbps, 40-CBR background.

background CBR traffic, and the number of CBR connections. We proposed and evaluated the effectiveness of a heuristic called ”fixed RTO.” With this heuristic, a TCP sender can determine if a retransmission timeout is due to network congestion or temporary route loss. In addition, we investigated the effectiveness of TCP’s selective and delayed acknowledgments in improving the performance.

Our simulations yield several interesting insights into the performances of the three algorithms. With standard Reno, the proactive ADV performs extremely well compared to the on-demand AODV and DSR. ADV provides lower connect times for TCP connections and higher throughputs as the number of TCP and CBR connections and volume of background traffic is varied. ADV’s routing overhead is generally lower in packets/s and higher in bits/s than that of AODV and DSR.

All three algorithms do not perform well when a TCP sender and receiver lose their route frequently and any new route established is likely to be longer than the old one. (See Figures 5 - 7 and the corresponding discussion given above.) Owing to its adaptive triggering of route updates, however, ADV suffers the least and is able to repair its routes fairly quickly. On the other hand, AODV and DSR perform poorly, since their route discovery mechanisms cannot handle such situations. This underscores the need to have some sort of proactive mechanism to repair routes quickly.

To improve the performances of the three algorithms, we used TCP’s selective and delayed acknowledgments, which yielded marginal performance gains for each of the three algorithms. Our proposed fixed-RTO mechanism improved the performances of the on-demand algorithms significantly, however. Since the retransmit timer is frozen and not doubled in cases where packet losses are due to broken routes, TCP retransmits a data packet more frequently. This, in turn, stimulates route discovery often enough that the on-demand algorithms are able to re-establish broken routes. The performance gains for DSR are similar to those reported by Holland et al. [13], who use more complicated explicit link failure notifications. ADV does not benefit from the fixed-RTO mechanism, since its route repairs do not occur any faster. It is noteworthy that both AODV and DSR outperform ADV as the number of TCP connections is increased and the proposed fixed-RTO heuristic is added to TCP Reno.

As the number of TCP connections (and, hence, the offered TCP traffic) is increased, while keeping the CBR background traffic fixed, ADV delivers significantly more CBR packets than AODV and DSR. This appears to indicate that ADV is likely to deliver real-time multimedia traffic reliably even in the presence of very large TCP traffic. (In our simulations, the offered TCP traffic ranged from 2-10 times that of CBR traffic.) However, we need to investigate this further.

Compared to the literature [1, 7, 13], this paper provides extensive analyses of routing algorithms and TCP performance enhancement mechanisms. In addition, our simulations indicate the relative strengths and weaknesses of on-demand algorithms that depend on route discovery mechanisms to repair broken routes and proactive algorithms that adaptively

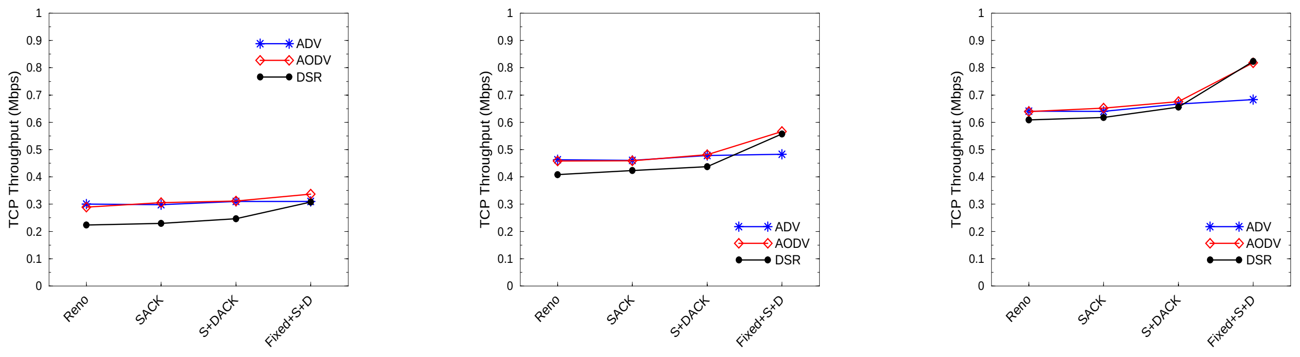

Figure 10: Combined throughputs for 2, 5, and 10 TCP connections with a 200 Kbps, 10-CBR background.

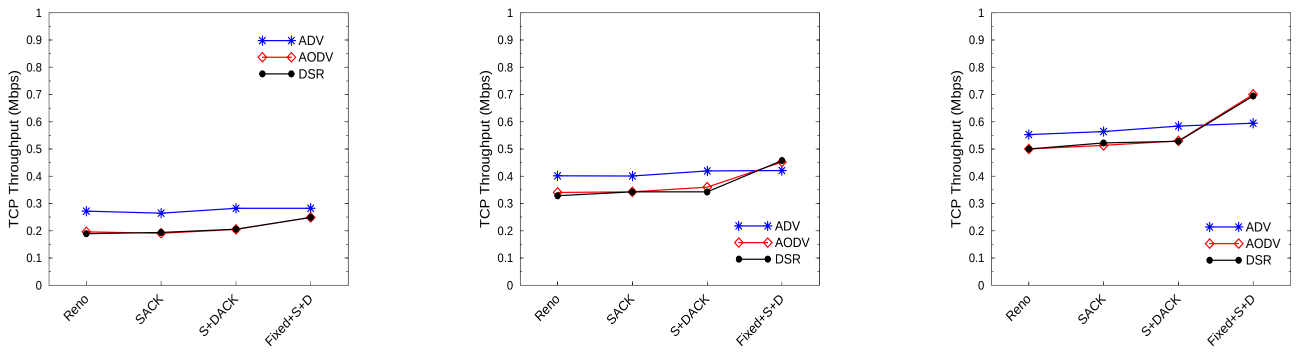

Figure 11: Combined throughputs for 2, 5, and 10 TCP connections with a 200 Kbps, 40-CBR background.

trigger route updates to repair broken routes. Our study also suggests that the fixed-RTO mechanism gives performance improvements comparable to those reported using explicit link failure notification (ELFN). We would like to simulate the ELFN scheme and compare it with the fixed-RTO mechanism to provide a more definitive conclusion.

In the future, we plan to enhance our study by incorporating HTTP traffic, where several TCP connections are opened and closed in short intervals. Given that ADV provides the shortest connection times, and that AODV and DSR give slightly higher throughputs with the fixed-RTO mechanism, it is not clear which algorithm will handle the HTTP traffic well. We would like to evaluate the algorithms for their ability to deliver real-time multimedia traffic in the presence of FTP and HTTP traffic.

7. REFERENCES

[1] A. Ahuja et al., "Performance of TCP over different routing protocols in mobile ad-hoc networks," Proceedings of IEEE Vehicular Technology Conference (VTC 2000), Tokyo, Japan, May 2000.

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

[3] H. Balakrishnan et al., "Improving reliable transport and handoff performance in cellular wireless networks," in *ACM SIGCOMM*, Aug. 1996.

[4] H. Balakrishnan et al., "A comparison of mechanisms for improving TCP performance over wireless links," *ACM SIGCOMM*, Aug. 1996.

[5] R. Boppana and S. Konduru, "An adaptive distance vector routing algorithm for mobile, ad hoc networks," in *IEEE Infocom 2001*, Mar. 2001.

[6] J. Broch et al., " A performance comparison of multi-hop wireless ad hoc network routing protocols" in *ACM Mobicom '98*, Oct. 1998.

[7] K. Chandran et al., "A feedback based scheme for improving TCP performance in ad-hoc wireless networks," in *Proc. International Conference on Distributed Computing Systems*, 1998.

[8] CMU Monarch Group, "CMU Monarch extensions to the NS-2 simulator." Available from http://monarch.cs.cmu.edu/cmu-ns.html, 1998.

[9] S. R. Das, C. E. Perkins, and E. M. Royer, "Performance comparison of two on-demand routing protocols for ad hoc networks," in *IEEE Infocom 2000*, Mar. 2000.

[10] R. Dube et al., "Signal stability based adaptive routing (SSA) for ad-hoc mobile networks," in *IEEE Personal Communications*, Feb. 1997.

[11] K. Fall and K. Varadhan, "NS notes and documentation." The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC. Available from http://www-mash.cs.berkeley.edu/ns, Nov. 1997.

[12] C. Hedrick, "Routing information protocol." RFC 1058, 1988.

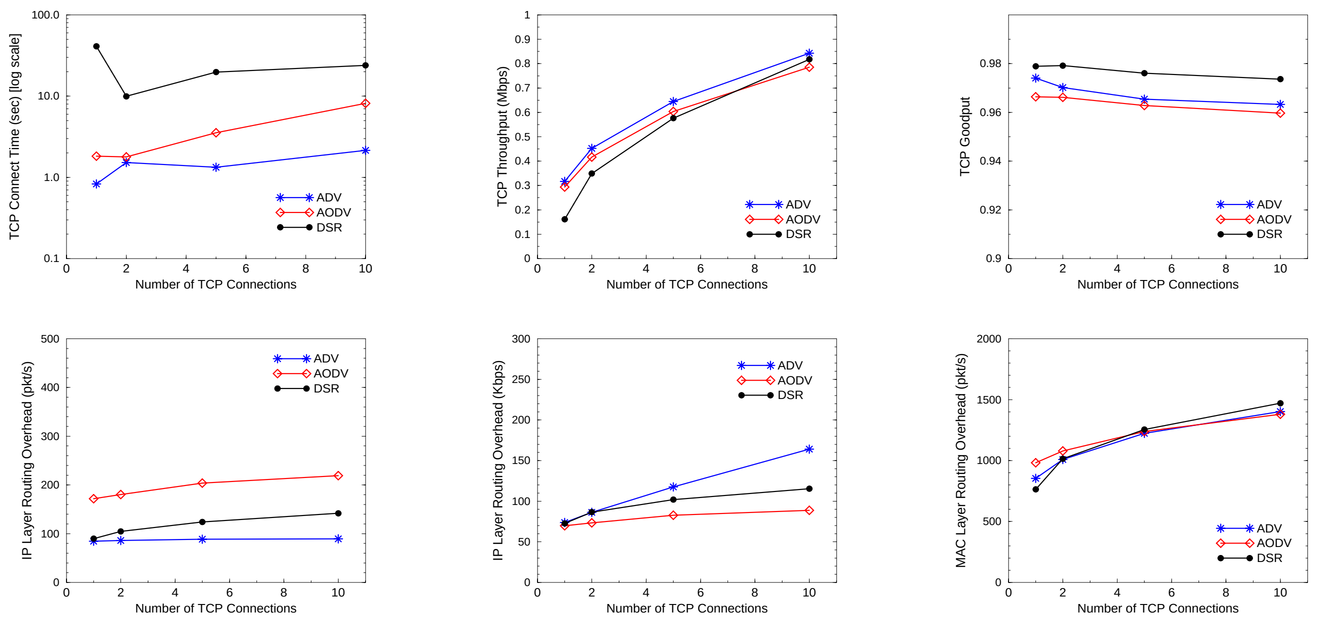

Figure 12: Connect times, throughputs, goodputs, and routing overhead for TCP Reno with a 100 Kbps 10-CBR background.

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

[14] IEEE Computer Society LAN / MAN Standards Committee, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications." IEEE Std. 802.11-1997. IEEE, New York, NY 1997.

[15] P. Johansson et al., "Scenario-based performance analysis of routing protocols for mobile ad-hoc networks," in *ACM Mobicom '99*, Aug. 1999.

[16] D. B. Johnson et al., "The dynamic source routing protocol for mobile adhoc networks." IETF Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-02.txt,
1999.

[17] S.Lee, Mario Gerla, and C.K.Toh, "A simulation study of table-driven and on-demand routing protocols for mobile ad hoc networks," in *IEEE Network Magazine*, Aug. 1999.

[18] C. E. Perkins, E. M. Royer, and S. R. Das, "Ad hoc on demand distance vector routing." IETF Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-03.txt,
1999.

[19] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance vector (DSDV) for mobile computers," in *ACM SIGCOMM '94*, pp. 234–244, Aug. 1994.

[20] N. Vaidya et al., "Delayed duplicate acknowledgements: a TCP-unaware approach to improve performance of TCP over wireless" Technical Report 99-003, Dept. of computer Science, Texas A&M University, Feb. 1999.

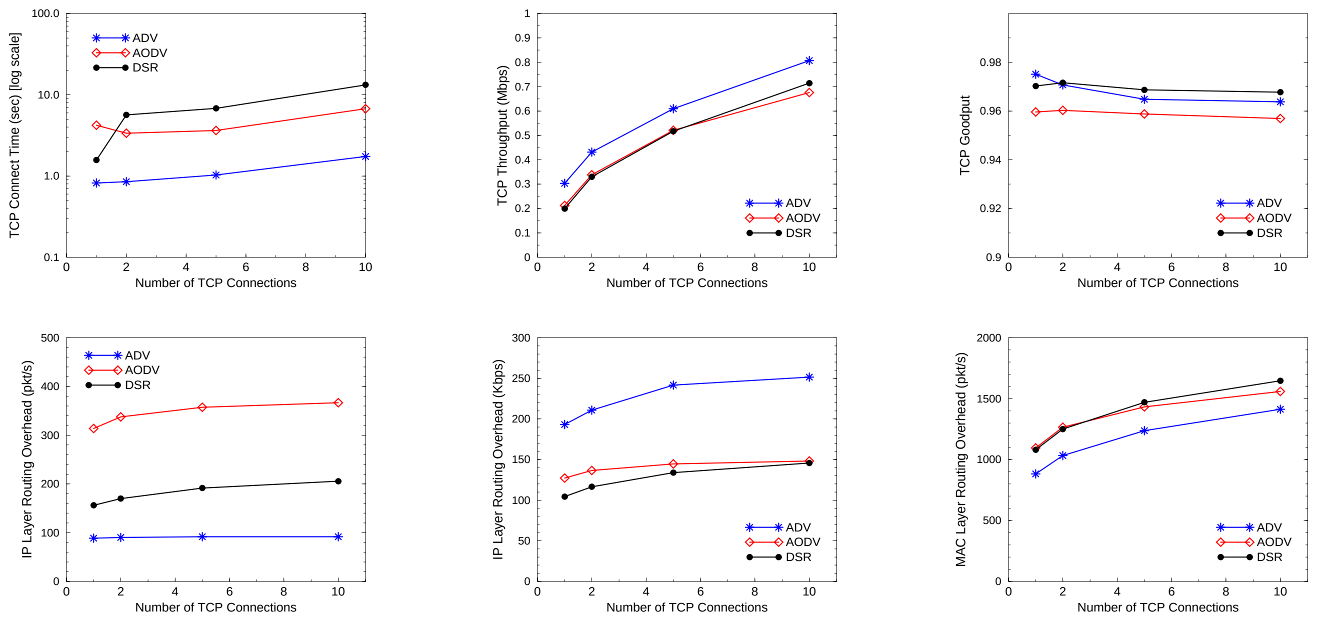

Figure 13: Connect times, throughputs, goodputs, and routing overhead for TCP Reno with a 100 Kbps 40-CBR background.

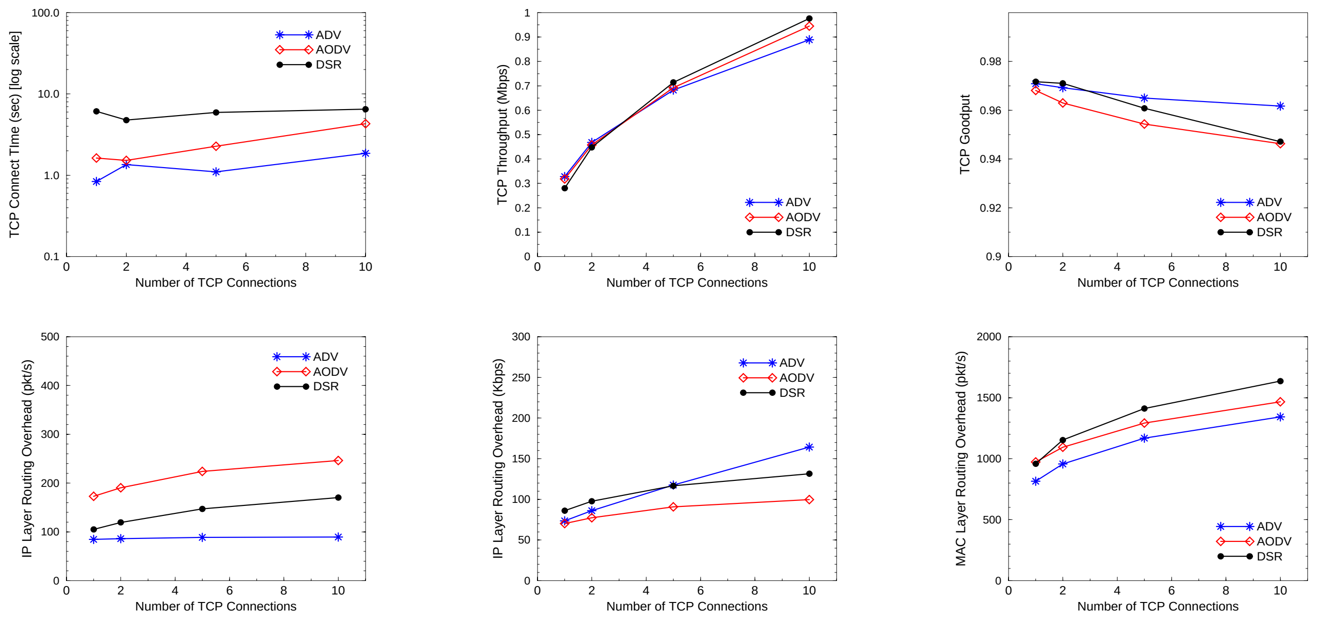

Figure 14: Connect times, throughputs, goodputs, and routing overhead for TCP SACK + delayed ACKs + fixed RTO with a 100 Kbps 10-CBR background.

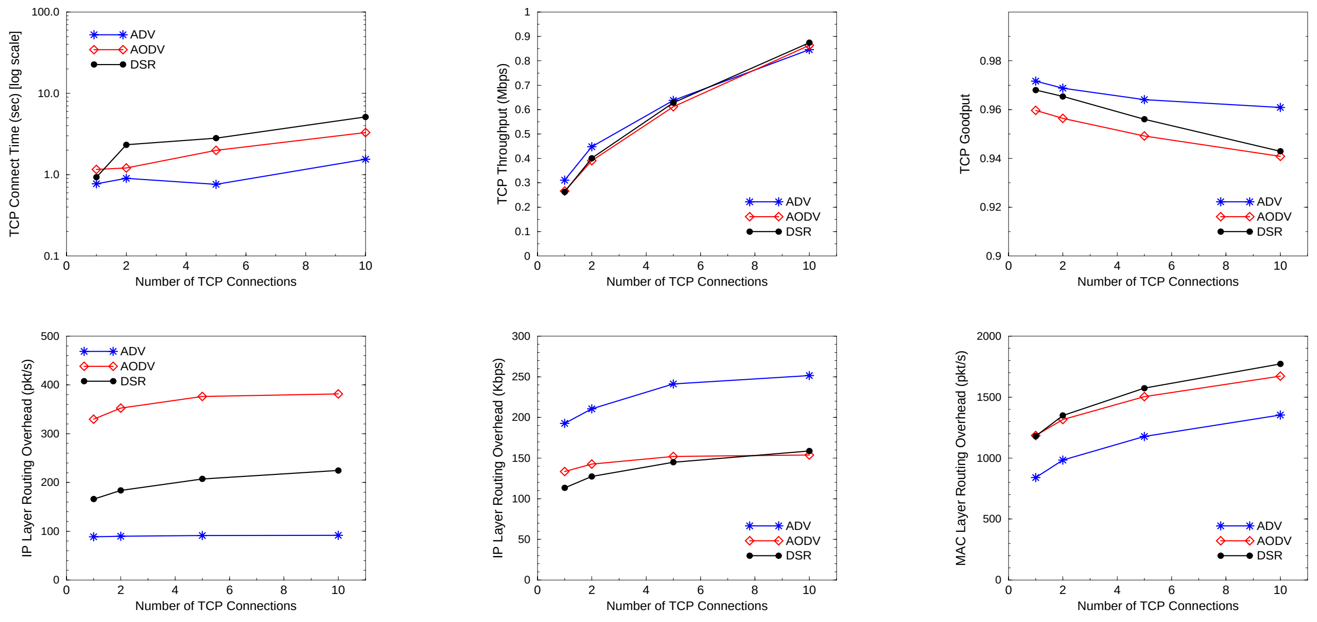

Figure 15: Connect times, throughputs, goodputs, and routing overhead for TCP SACK + delayed ACKs + fixed RTO with a 100 Kbps 40-CBR background.

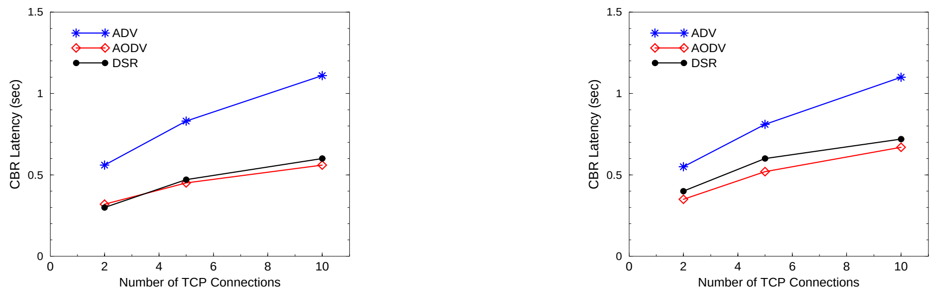

Figure 16: CBR latencies for TCP Reno and TCP SACK + delayed ACKs + fixed RTO with a 100 Kbps 10-CBR background.

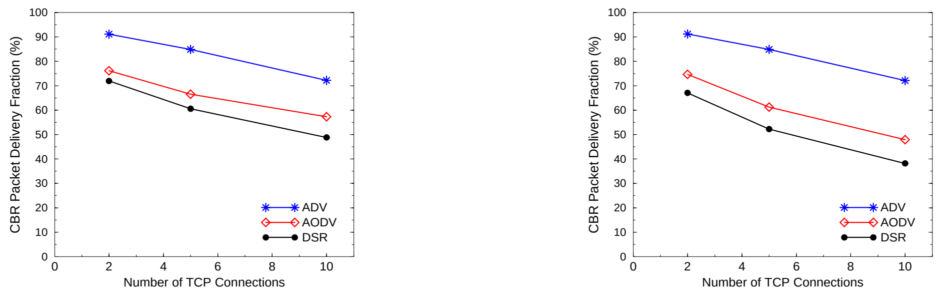

Figure 17: CBR packet delivery fractions for TCP Reno and TCP SACK + delayed ACKs + fixed RTO with a 100 Kbps 10-CBR background.