Websites:
http://www.sciencepub.net
http://www.sciencepub.net/rural

Emails:
editor@sciencepub.net
sciencepub@gmail.com

World Rural Observations

MARSLAND PRESS
Multidisciplinary Academic Journal Publisher

A Simulations Based Study for Diagnosing Impacts of MANETs Routing Protocols on TCP Performance with Background Traffic

Salman Faiz Solehria[1], Salim-ur-Rehman[2], Shaukat Ali[3]

[1,2]Department of Computer Science, Sarhad University of Science & Information Technology Peshawar, Pakistan
[3]Department of Computer Science, FG Degree Peshawar Cantt, Pakistan
{salman, salimurrehman }@suit.edu.pk, shoonikhan@gmail.com, salman@suit.edu.pk

Abstract: TCP is the major component of computer network whether it's wired or wireless. The standard TCP was developed for wired network. In Mobile Ad Hoc Network (MANETs) legacy TCP performs poorly and require drastic changes. All most, all TCP performance studies are based on such simulations in which the real world problems are ignored such as background traffic or other interference caused by WiFi hotspot and other devices in range. In order to delve the effect of background traffic on the relative TCP performance, a simulation has been developed to model the back ground traffic. This paper studies the effect of routing protocols on TCP performance with background traffic. Three different routing protocols (AODV, DSDV, and DSR) are evaluated with three different TCP variants (SACK, NewReno, Tahoe). The performances parameters that are calculated in this paper are throughput, congestion window and end to end delay etc. It was deduced from the simulation results that SACK TCP produces 6.66 % throughput with DSR while no other combination produces more than 0.66 % throughput.

[Salman Faiz Solehria, Salim-ur-Rehman, Shaukat Ali. **A Simulations Based Study for Diagnosing Impacts of MANETs Routing Protocols on TCP Performance with Background Traffic**. World Rural Observ 2021;13(4):49-59]. ISSN: 1944-6543 (Print); ISSN: 1944-6551 (Online). http://www.sciencepub.net/rural. 5. doi:10.7537/marswro130421.05.

Keywords: TCP; MANETs; AODC; DSDV; DSR; SACK; NewReno; Tahoe

1. Introduction

Due to the rapid growth in technology Mobile Ad hoc network is becoming part of life day by day. Mobile Ad-hoc Networks (MANETS) exchange information in a wireless environment between a numbers of autonomous nodes and forming a multi hop radio network without any pre-existing infrastructure and base station i.e. the nodes communicates directly with one another in a peer-to-peer fashion. The incredible growth of private computers and the clever usage of mobile computers necessitate the need to sharing of information among computers. Currently this sharing of information is complex, as the users need to perform administrative tasks and set up static, bi-directional links between the computers. This motivates the construction of temporary networks with no wires and no communication infrastructure and no administrative intervention required. Such an interconnection between mobile computers is called an Ad-hoc Network. In such a setting, it may be necessary for the mobile computers to take help of other computers in forwarding a packet to the destination due to the limited scope of each Mobile host's wireless transmission.

2. Relative Work

Although some related work has been carried out in (Bakht, 2004) (Das et al., 2000) (Dyer and Boppana, 2001) (Johansson et al., 1999) (Pucha et al., 2004) (Trung et al., 2007) but they have encountered other factors (i.e. node density, mobility, node pause times traffic, delay, large topology area etc) that can effect TCP performance significantly and have not justified the actual affect of the routing protocols on TCP performance. A closely related work is carried out in (Fall and Floyd, 1996), but it is based on the fixed wired networks not on Mobile Ad-Hoc Networks. Another closest work has been done by (Pucha et al., 2004) but they have changed node density and node pause times and have not touched background traffic.

3. Routing Algorithm

We have considered mainly three routing algorithms i.e. DSDV, DSR and AODV. These routing protocols are analyzed with different variants of TCP.

3.1. Destination Sequenced Distance Vector-DSDV

DSDV (Perkins and Bhagwat, 1994) is a hop-by-hop distance vector routing protocol that in each node has a routing table that for all reachable destinations stores the next-hop and number of hops for that destination. Like distance-vector, DSDV requires that each node periodically broadcast routing updates. The advantage with DSDV over traditional distance vector protocols is that DSDV guarantees loop-freedom. To guarantee loop-freedom DSDV uses a sequence numbers to tag each route. The sequence number shows the freshness of a route and routes with higher sequence numbers are favorable. A route R is considered more favorable than R' if R has a greater sequence number or, if the routes have the same sequence number but R has lower hop-count. The sequence number is increased when a node A detects that a route to a destination D has broken. So the next time node A advertises its routes, it will advertise the route to D with an infinite hop-count and a sequence number that is larger than before. DSDV basically is distance vector with small adjustments to make it better suited for ad-hoc networks. These adjustments consist of triggered updates that will take care of topology changes in the time between broadcasts. To reduce the amount of information in these packets there are two types of update messages defined, full and incremental dump. The full dump carries all available routing information and the incremental dump that only carries the information that has changed since the last dump.

3.2. Ad-hoc On Demand Distance Vector-AODV

The Ad Hoc On-Demand Distance Vector (AODV) (Perkins and Royer, 1999) routing protocol enables multi-hop routing between participating mobile nodes wishing to establish and maintain an ad-hoc network. AODV is based upon the distance vector algorithm. The difference is that AODV is reactive, as opposed to proactive protocols like DV, i.e. AODV only requests a route when needed and does not require nodes to maintain routes to destinations that are not actively used in communications. As long as the endpoints of a communication connection have valid routes to each other, AODV does not play any role. Features of this protocol include loop freedom and that link breakages cause immediate notifications to be sent to the affected set of nodes, but only that set. Additionally, AODV has support for multicast routing and avoids the Bellman Ford "counting to infinity" problem (Steenstrup, 1995). The use of destination sequence numbers guarantees that a route is "fresh". The algorithm uses different messages to discover and maintain links. Whenever a node wants to try and find a route to another node, it broadcasts a Route Request (RREQ) to all its neighbors. The RREQ propagates through the network until it reaches the destination or a node with a fresh enough route to the destination. Then the route is made available by unicasting a RREP back to the source.

The algorithm uses hello messages (a special RREP) that are broadcasted periodically to the immediate neighbors. These hello messages are local advertisements for the continued presence of the node and neighbors using routes through the broadcasting node will continue to mark the routes as valid. If hello messages stop coming from a particular node, the neighbor can assume that the node has moved away and mark that link to the node as broken and notify the affected set of nodes by sending a link failure notification (a special RREP) to that set of nodes.

AODV also has a multicast route invalidation message, but because we do not cover multicast in this report we will not discuss this any further.

For routing table management, AODV needs to keep track of the following information for each route table entry:

1. Destination IP Address: IP address for the destination node.
2. Destination Sequence Number: Sequence number for this destination.
3. Hop Count: Number of hops to the destination.
4. Next Hop: The neighbor, which has been designated to forward packets to the destination for this route entry.
5. Lifetime: The time for which the route is considered valid.
6. Active neighbor list: Neighbor nodes that are actively using this route entry.
7. Request buffer: Makes sure that a request is only processed once.

For route discovery, a node broadcasts a RREQ when it needs a route to a destination and does not have one available. This can happen if the route to the destination is unknown, or if a previously valid route expires. After broadcasting a RREQ, the node waits for a RREP. If the reply is not received within a certain time, the node may rebroadcast the RREQ or assume that there is no route to the destination. Forwarding of RREQs is done when the node receiving a RREQ does not have a route to the destination. It then rebroadcast the RREQ. The node also creates a temporary reverse route to the Source IP Address in its routing table with next hop equal to the IP address field of the neighboring node that sent the broadcast RREQ. This is done to keep track of a route back to the original node making the request, and might be used for an eventual RREP to find its way back to the requesting node. The route is temporary in the sense that it is valid for a much shorter time, than an actual route entry. When the RREQ reaches a node that either is the destination node or a node with a valid route to the destination, a RREP is generated and unicasted back to the requesting node. While this RREP is forwarded, a route is created to the destination and when the RREP reaches the source node, there exists a route from the source to the destination.

For route maintenance, when a node detects that a route to a neighbor no longer is valid, it will remove the routing entry and send a link failure message, a triggered route reply message to the neighbors that are actively using the route, informing them that this route no longer is valid. For this purpose AODV uses a active neighbor list to keep track of the neighbors that are using a particular route. The nodes that receive this message will repeat this procedure. The message will eventually be received by the affected sources that can chose to either stop sending data or requesting a new route by sending out a new RREQ.

3.3. Dynamic Source Routing - DSR

Dynamic Source Routing (DSR) (Broch et al., 1998) (Johnson and Maltz, 1996) (Johnson and Maltz, 1996) also belongs to the class of reactive protocols and allows nodes to dynamically discover a route across multiple network hops to any destination. Source routing means that each packet in its header carries the complete ordered list of nodes through which the packet must pass. DSR uses no periodic routing messages (e.g. no router advertisements), thereby reducing network bandwidth overhead, conserving battery power and avoiding large routing updates throughout the ad-hoc network. Instead DSR relies on support from the MAC layer (the MAC layer should inform the routing protocol about link failures). The two basic modes of operation in DSR are route discovery and route maintenance.

Route discovery: Route discovery is the mechanism whereby a node X wishing to send a packet to Y, obtains the source route to Y. Node X requests a route by broadcasting a Route Request (RREQ) packet. Every node receiving this RREQ searches through its route cache for a route to the requested destination. DSR stores all known routes in its route cache. If no route is found, it forwards the RREQ further and adds its own address to the recorded hop sequence. This request propagates through the network until either the destination or a node with a route to the destination is reached. When this happen a Route Reply (RREP) is unicasted back to the originator. This RREP packet contains the sequence of network hops through which it may reach the target. In Route Discovery, a node first sends a RREQ with the maximum propagation limit (hop limit) set to zero, prohibiting its neighbors from rebroadcasting it. At the cost of a single broadcast packet, this mechanism allows a node to query the route caches of all its neighbors. Nodes can also operate their network interface in promiscuous mode, disabling the interface address filtering and causing the network protocol to receive all packets that the interface overhears. These packets are scanned for useful source routes or route error messages and then discarded. The route back to the originator can be retrieved in several ways. The simplest way is to reverse the hop record in the packet. However this assumes symmetrical links. To deal with this, DSR checks the route cache of the replying node. If a route is found, it is used instead. Another way is to piggyback the reply on a RREQ targeted at the originator. This means that DSR can compute correct routes in the presence of asymmetric (unidirectional) links. Once a route is found, it is stored in the cache with a time stamp and the route maintenance phase begins.

Route maintenance is the mechanism by which a packet sender S detects if the network topology has changed so that it can no longer use its route to the destination D. This might happen because a host listed in a source route, move out of wireless transmission range or is turned off making the route unusable. A failed link is detected by either actively monitoring acknowledgements or passively by running in promiscuous mode, overhearing that a packet is forwarded by a neighboring node. When route maintenance detects a problem with a route in use, a route error packet is sent back to the source node. When this error packet is received, the hop in error is removed from this hosts route cache, and all routes that contain this hop are truncated at this point.

4. Transmission Control Protocol-TCP

Although TCP is free from the underlying network technologies, some assumptions in its characteristics are clearly inspired for wired networks dominant at the time when it was conceived. TCP implicitly assume that modes are static (i.e. they do not change their position over time) and packet loss is due to congestion causing buffer overflows at the intermediate nodes (routers).This assumption does not hold in MANETs as the network topology may change due to node movement and failure (e.g. because the battery is exhausted). Packet losses due to buffer overflow are rare events in MANET. Therefore the legacy TCP performs badly in MANETs.

4.1. Tahoe TCP

Tahoe by Jacobson (Jacobson, 1995) assumed that congestion signals are represented by lost segments. It was assumed by Jacobson that losses due to packet corruption are much less probable than losses due to buffer overflows on the network. Therefore, on a loss, the sender should lower its share of the bandwidth. Tahoe TCP includes Slow-Start, Congestion Avoidance and Fast Retransmit. Tahoe TCP detect packet loss by either expiring RTO or receiving 3 duplicate ACKs. In case of receiving 3 duplicate ACKs Tahoe TCP retransmits the lost packet without having to wait for the RTO to expire. As RTO is relatively quite enough to transmit a packet, this process is called Fast Retransmit. Upon receiving a congestion signal Tahoe

TCP sets its threshold value to half of the value of the congestion window just before the lost is recorded and congestion window is set to 1 MSS and enters into slow start. This process is called congestion avoidance. Tahoe TCP does not deal well with multiple packet loss within a single window of data.

4.2. Reno TCP

Reno TCP introduced a major improvement over Tahoe TCP by detecting packet loss via time out and not via duplicate ACKs. Upon receiving duplicate ACKs, Reno TCP does not enter into Slow-Start but into Fast Recovery, which should be activated after Fast Retransmit.

In Fast Recovery, threshold value is set to half of the congestion window. The congestion window is set to threshold plus 3 * MSS (Where 3 is the number of duplicate ACKs required to trigger Fast Retransmit). Upon the receipt of each duplicate ACK, the congestion window is inflated by one segment as each duplicate ACK signals the fact that another packet has left the network. After receiving W/2 duplicate ACKs (where W is the size of congestion window before receiving the duplicate ACKs), the window will be ready to include new packets. When all of the duplicate ACKs are received, the congestion window is set to threshold value (W/2). After this congestion window is increased one packet at a time. In case of RTO Reno TCP exhibits the same behavior as Tahoe TCP. Although Reno TCP is better than Tahoe TCP in dealing with single packet loss, Reno TCP is not better when multiple packets are lost within a single window of data (Jacobson, 1995). In the Internet, packets are often transmitted in bursts (Comer and Stevens, 1994). As a result, losses often happen in bursts. This is primarily due to FIFO (Drop Tail) queues in routers. Therefore Reno TCP faces problems because of its Fast Recovery and Fast Retransmit (which can lead to impacting throughput of the connection). Two solutions are provided to overcome Reno TCP multiple packets loss problem: NewReno TCP and SACK TCP.

4.3. NewReno TCP

NewReno modifies the Fast Retransmit and Fast Recovery. These modifications are intended to fix the Reno problems above and are wholly implemented in the sender side. A modification of Reno lead to NewReno TCP which shows that Reno can be improved without the addition of SACKs but still suffers without it. Here, the wait for a retransmit timer is eliminated when multiple packets are lost from a window. NewReno is the same as Reno but with more intelligence during Fast Recovery (Clark and Hoe, 1995) (Hoe, 1995). It utilizes the idea of partial ACKs: when there are multiple packet drops, the ACKs for the retransmitted packet will acknowledge some, but not all the segments send before the Fast Retransmit. In TCP Reno, the first partial ACK will bring the sender out of the Fast Recovery phase. This will result in the requirement of timeouts when there are multiple losses in a window, and thus stalling the TCP connection.

In NewReno, a partial ACK is taken as an indication of another lost packet and as such the sender retransmits the first unacknowledged packet. Unlike Reno, partial ACKs don't take NewReno out of Fast Recovery. This way, it retransmits one packet per RTT until all the lost packets are retransmitted and avoids requiring multiple Fast Retransmits from a single window of data. The downside of this is that it may take many RTT's to recover from a loss episode, and you must have enough new data around to keep the ACK clock running. This is implemented as follows:

- Multiple Packet Loss: A fix for Fast Recovery to prevent starting Fast Retransmit and Fast Recovery in succession when multiple segments are dropped in the same window. When entering Fast Retransmit (from 3 duplicate ACKs), have the highest sequence number sent so far. Perform retransmission and the Fast Recovery algorithm as usual (set threshold, inflating congestion window on duplicate ACKs). When a new ACK arrives, perform the addition check if the ACK covers the highest sequence number sent when Fast Retransmit was invoked. If not, this ACK is a partial ACK and signals that another segment was lost from the same window of data. As such, retransmit the segment reported as expected by the partial ACK, reset the retransmission timer but do not exit Fast Recovery. On the other hand, if the new ACK covers the highest sequence number sent and then exit Fast Recovery but setting congestion window to threshold and performing congestion avoidance.
- False Fast Retransmit: Record the highest sequence number ever transmitted after a retransmission timeout (normally set to 0). Whenever we get 3 duplicate ACKs, we perform a test to see if we should enter Fast Retransmit. If these ACKs cover the sequence number saved at the previous timeout, then this is a new invocation of the Fast Retransmit. In this case, enter Fast Retransmit and perform the related actions. If they do not cover the sequence numbers (i.e. they ACK segments sent previous to the timeout) then just acknowledge the receipt of already queued segments at the receiver. In this case, do not enter Fast Retransmit. Sender comes out of Fast Recovery only after all outstanding packets (at the time of first loss) are ACKed.

4.4. SACK TCP

The SACK option follows the format in the RFC2018. The SACK option field contends a number of SACK blocks, where each SACK block report a non-contiguous set of data that has been received and queued. The first block in a SACK option is required to report the data receiver's most recently received segment, and the additional SACK blocks repeat the most recently reported SACK blocks.

The congestion control algorithms implemented in our SACK TCP area conservative extension of Reno's congestion control, in that they use the same algorithms for increasing and decreasing the congestion window, and make minimal changes to the other congestion control algorithms. Adding SACK to TCP does not change the basic underlying congestion control algorithms. The SACK TCP implementation preserves the properties of Tahoe and Reno TCP of being robust in the presence of out-of-order packets, and uses retransmit timeouts as the recovery method of last resort. The main difference between the SACK TCP implementation and the Reno TCP implementation is in the behavior when multiple packets are dropped from one window of data.

As in Reno, the SACK TCP implementation enters Fast Recovery when the data sender receives duplicate acknowledgements. The sender retransmits a packet and cut the congestion window in half. During Fast Recovery, SACK maintains a variable called "pipe" that represents the estimated number of packets outstanding in the path; this differs from the implementation in the Reno implementation. The sender only sends new or retransmitted data when the estimated number of packets in the path is less than the congestion window. The variable "pipe" is incremented by one when the sender either sends a new packet or retransmits an old packet. It is decremented by one when the sender receives a duplicate ACK packet with a SACK option reporting the new data has been received at the receiver. Use of the "pipe" variables decouples the decision of when to send a packet from the decision of which packet to send. The sender maintains a data structure, the scoreboard that remembers acknowledgement from previous SACK option. When the sender is allowed a packet, it retransmits the next packet from the list of packets inferred to be missing at the receiver. If there are no such packets and the receiver's advertised window is sufficiently large, the sender sends a new packet. When a retransmitted packet is itself dropped, the SACK implementation detects the drop with a retransmit timeout, retransmitting the dropped packet and then Slow-Starting. The sender exits Fast Recovery when a recovery acknowledgement is received acknowledging all data that was outstanding when Fast Recovery was entered.

The SACK sender has special handling for partial ACKs (ACKs received during Fast Recovery that advance the Acknowledgement Number field of the TCP header, but do not take the sender out of Fast Recovery). For partial ACKs, the sender decrements "pipe" by two packets rather than one, as follows. When Fast Retransmit is initiated, "pipe" is effectively decremented by one for the packet that was assumed to have been dropped, and then incremented by one for the packet that was retransmitted. Thus, decrementing the "pipe" by two packets when the first partial ACK is received is in some sense "cheating", as that partial ACK only represents one packet having left the pipe. However, for any succeeding partial ACKs, "pipe" was incremented when the retransmitted packet entered the pipe, but was never decremented for the packet assumed to have been dropped. Thus, when the succeeding partial ACK arrives, it does in fact represent two packets that have left the "pipe": the original packet (assumed to have been dropped), and the retransmitted packet. Because the sender decrements "pipe" by two packets rather than one for partial ACKs, the SACK sender never recovers more slowly than a Slow-Start. The "maxburst" parameter, which limits the number of packets that can be sent in response to a single incoming ACK packet, is experimental, and is not necessarily recommended for SACK implementation.

There are a number of other proposals for TCP congestion control algorithm using selective acknowledgements (Mathis & Mahdavi, 1996). The SACK implementation in our simulator is designed to be the most conservative extension of the Reno congestion control algorithms, in that it makes the minimum changes to Reno's existing congestion control algorithms.

5. Simulations Environment

The simulator we have used to simulate the Mobile Ad-hoc Networks routing protocols over TCP is the Network Simulator 2 (NS-2) from Berkeley. To simulate the mobile wireless radio environment we have used a mobility extension to NS that is developed by the CMU Monarch project at Carnegie Mellon University. NS-2 along with CMU-extension provides support for all TCP implementation and routing protocols (DSDV, DSR and AODV) that we have used in this research paper. All of these protocols are used with their basic configurations and no changes have been made.

5.1 Simulation Objective

In this paper we are interested in finding out that how background traffic effects the relative performance of TCP. The traffic generated by the source node is not the only traffic which can create congestion and can affect TCP and routing protocols performance. Selected routing protocols (DSDV DSR and AODV) are simulated with different TCP variants

(Tahoe, NewReno and SACK). TCP performance can be measured in TCP congestion window, TCP throughput, number of packets sent, number of acknowledgement received, round trip time etc. We have collected data about these parameters and concluded created my results.

5.2 Simulation Scenario

My scenario consists of three wireless nodes over the area of a size of 500m x 400m. A TCP connection is established between the node(0) and node(1) with having background traffic. The initial location of the node(0), node(1) and node(2) as shown in Figure 1, are receptively (5, 5), (490, 285) and (150, 240), whereas the z co-ordinate is assumed to be 0.

5.3. Traffic Pattern

The traffic pattern defines the way in which the mobile node moves in a scenario. The traffic pattern for the three mobile nodes in the simulation scenario is defined in such a way that they remain in each other's radio range for the maximum of simulation time, so that the TCP connection should be remained open. The traffic pattern is: at time 10, node(0) starts moving towards point (250, 250) at the speed of 3 m/sec, at time 15, node (1) starts moving towards point (48, 285) at a speed of 5 m/sec, at time 110, node(0) starts moving towards point (480, 300) at a speed of 5 m/sec and at time 110, node(1) starts moving towards point (10,150) at a speed of 5 m/sec. Node(2) is kept static in the simulation. It will just act as a relay between node(0) and node(1) to help in increasing the connection time and introduces an imaginary propagation and processing delay. The duration of the simulation is kept 150 seconds, as it is the maximum time to get the required results.

Figure 1. Initial nodes position.

6. Connection Pattern

The connection pattern defines that who is connected to whom in the simulation scenario. In simulation scenario a TCP connection is established between node(0) and node(1). The background traffic is discussed later on. Node(0) acts as TCP source and node(1) acts as a TCP sink. TCP source sends TCP packets and TCP sink sends acknowledgements back to the source. The TCP packet size is set to its original length (1000Kb). Node(0) sends FTP packets. At time 12.0 FTP transmission starts, but when node(0) and node(1) comes in each other radio range or in the radio range of node(2), TCP connection will be created and data transmission will take place.

6.1 Background Traffic

Most of the research work has ignored the real world problems and therefore they are less applicable to the real world environment. This simulation is designed with real world interference in mind caused by background traffic and other factors. Only background traffic is considered in this paper.

The background traffic consists of three UDP sessions communicating at a constant bit rate (CBR). The packet size is 512 bytes and the interval is 0.25 second so the total background traffic for 130 seconds of the simulation is 260kbps. At 15 second just after the FTP traffic starts at 12 second node(2) establish a UDP connection with node(0) and starts communicating up to 65 second at a constant bit rate. After this at 66 second node(2) initiates a UDP connection with node(1) up to 100 second. Similarly a third UDP connection is established between node(0) and node(1) from 101 to 149 second. The background traffic is designed in such a way to provide maximum interference to the communication nodes.

7. Results

In this section, we will discuss the results that have been achieved after running the simulation script written in OTCL. We have used different Mobile Ad-Hoc Networks routing protocol and TCP algorithms, therefore have discussed them one by one. The overall results obtained are shown in Table 4.

7.1. TCP Algorithms with DSDV

When DSDV is simulated with different TCP algorithms, the results obtained are show in Figure : 2. A detailed statistics of the number of packets lost, ACKs lost, and throughput in percentage is shown in the Table 1.

7.2. TCP algorithms with DSR:

When DSR was simulated with the three algorithms of TCP the results obtained are shown in Figure 3. A detailed statistics of the number of packets lost, ACKs lost, and throughput in percentage is shown in the Table 2.

7.3. TCP algorithms with AODV:

When AODV was simulated with the three algorithms of TCP the results obtained are shown in Figure 4. A detailed statistics of the number of packets lost, ACKs lost, and throughput in percentage is shown in the Table 3.

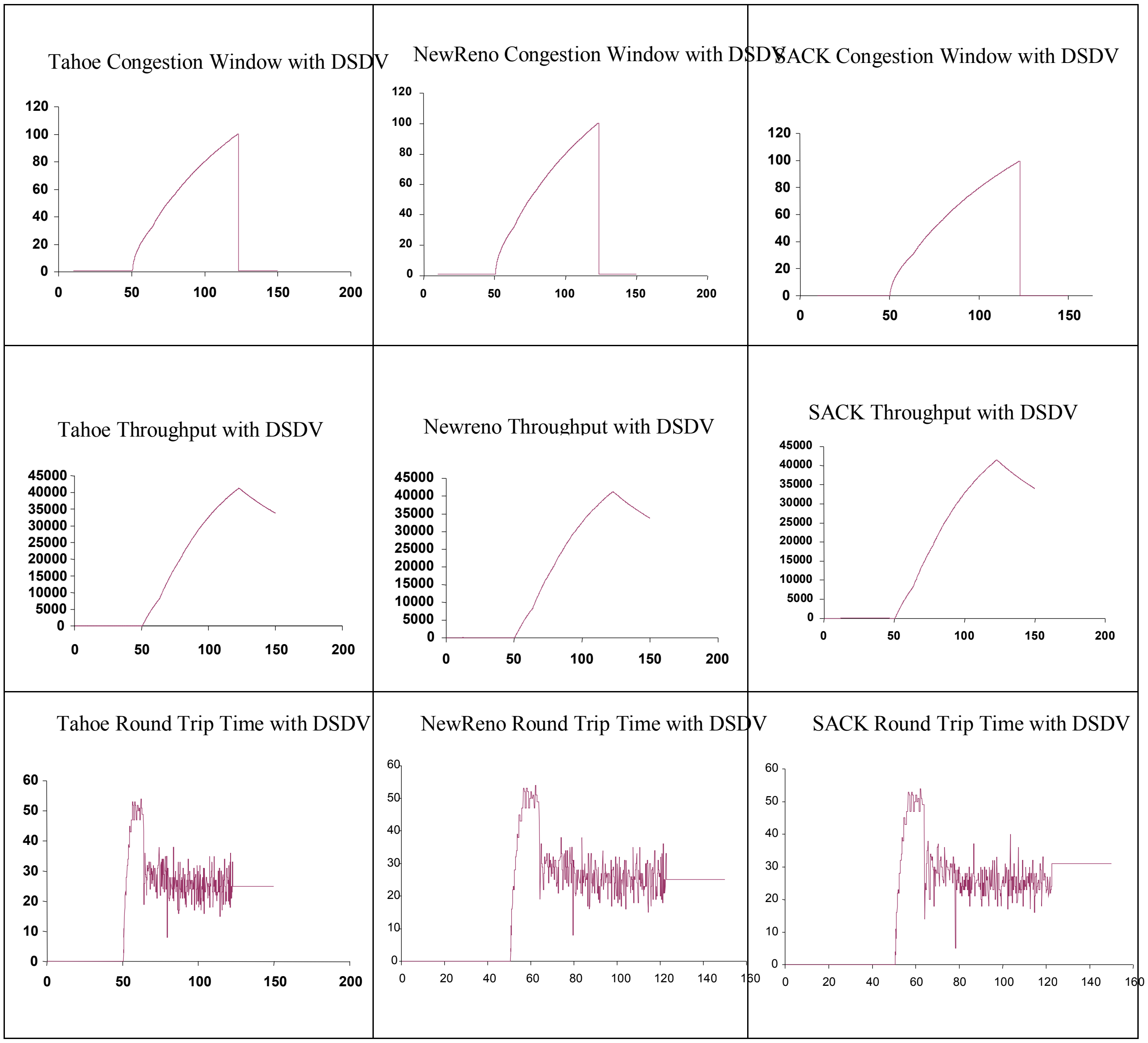

Figure 2. DSDV results with Tahoe, NewReno, and SACK.

Table 1. TCP algorithms with DSDV

S.No	Routing Protocol	TCP Variant	Transmitted Bytes	Acknowledged Packets	Throughput	Lost Packet	Throughput %
1	DSDV	Tahoe	5079000	5052	33835.89	7.0	0.666
2	DSDV	NewReno	5079000	5052	33835.89	7.0	0.666
3	DSDV	SACK	5105000	5077	34002.67	8.0	0.666

Table 2. TCP algorithms with DSR

S.No	Routing Protocol	TCP Variant	Transmitted Bytes	Ack_Pack	Throughput	Lost Packet	Throughput %
1	DSR	SACK	665000	6628	44322.88	6.0	6.66
2	DSR	Tahoe	6652000	6629	44269.51	16.0	0.665
3	DSR	Newreno	6649000	6626	44289.53	10.0	0.666

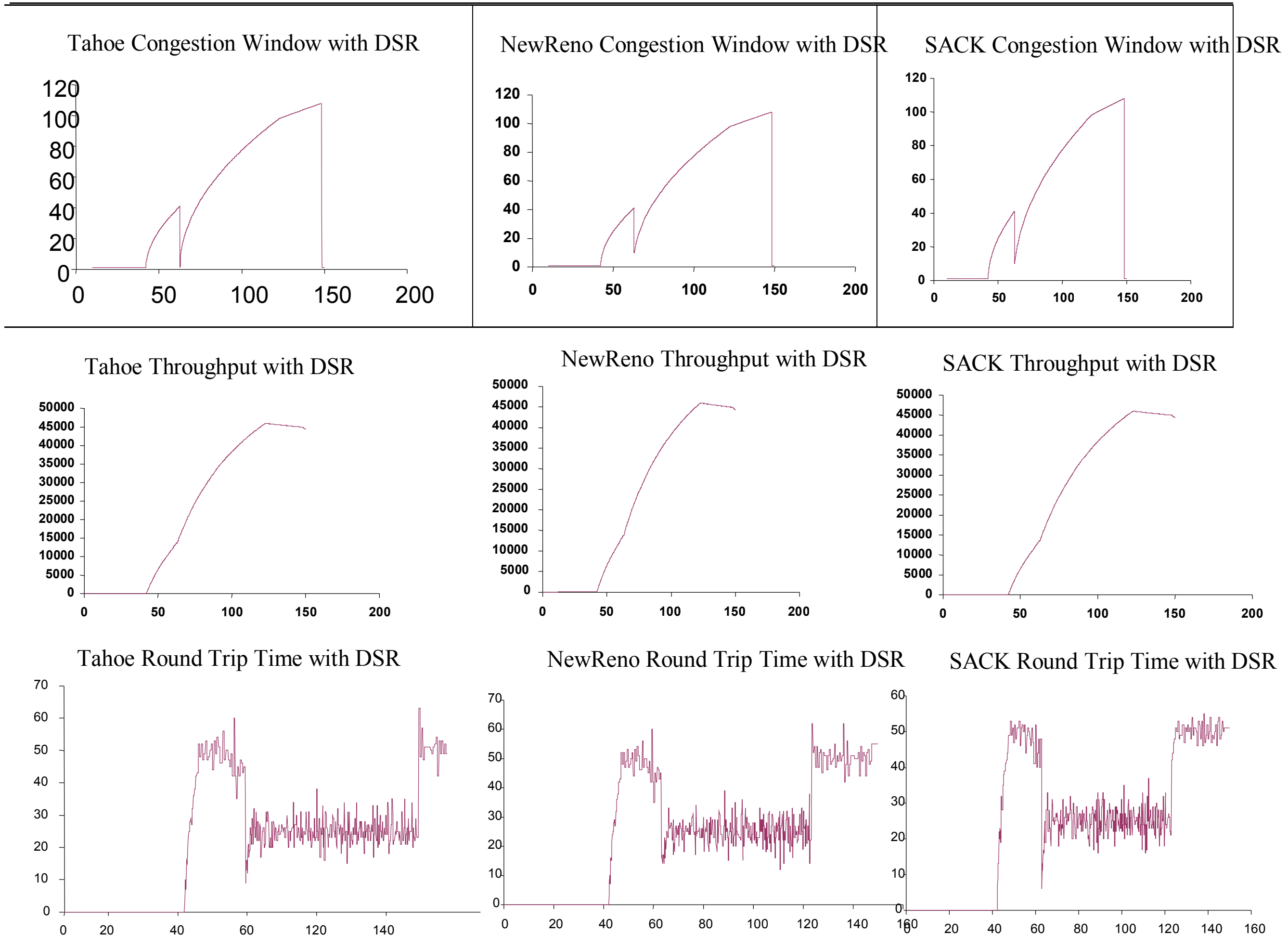

Figure 3. DSDV results with Tahoe, NewReno, and SACK.

Table 3. TCP algorithms with AODV

S.No	Routing Protocol	TCP Variant	Transmitted Bytes	Ack_Pack	Throughput	Lost Packet	Throughput %
1	AODV	Newreno	3764000	3739	25076.72	5.0	0.666
2	AODV	SACK	3764000	3739	25076.72	5.0	0.666
3	AODV	Tahoe	3764000	3739	25076.72	5.0	0.666

Table 4. Overall comparision

S.No	Routing Protocol	TCP Variant	Transmitted Bytes	Ack_Pack	Throughput	Lost Packet	Throughput %
1	DSR	SACK	665000	6628	44322.88	6.0	6.66
2	DSR	Tahoe	6652000	6629	44269.51	16.0	0.665
3	DSR	Newreno	6649000	6626	44289.53	10.0	0.666
4	DSDV	SACK	5105000	5077	34002.67	8.0	0.666
5	DSDV	Tahoe	5079000	5052	33835.89	7.0	0.666
6	DSDV	Newreno	5079000	5052	33835.89	7.0	0.666
7	AODV	SACK	3764000	3739	25076.72	5.0	0.666
8	AODV	Tahoe	3764000	3739	25076.72	5.0	0.666
9	AODV	Newreno	3764000	3739	25076.72	5.0	0.666

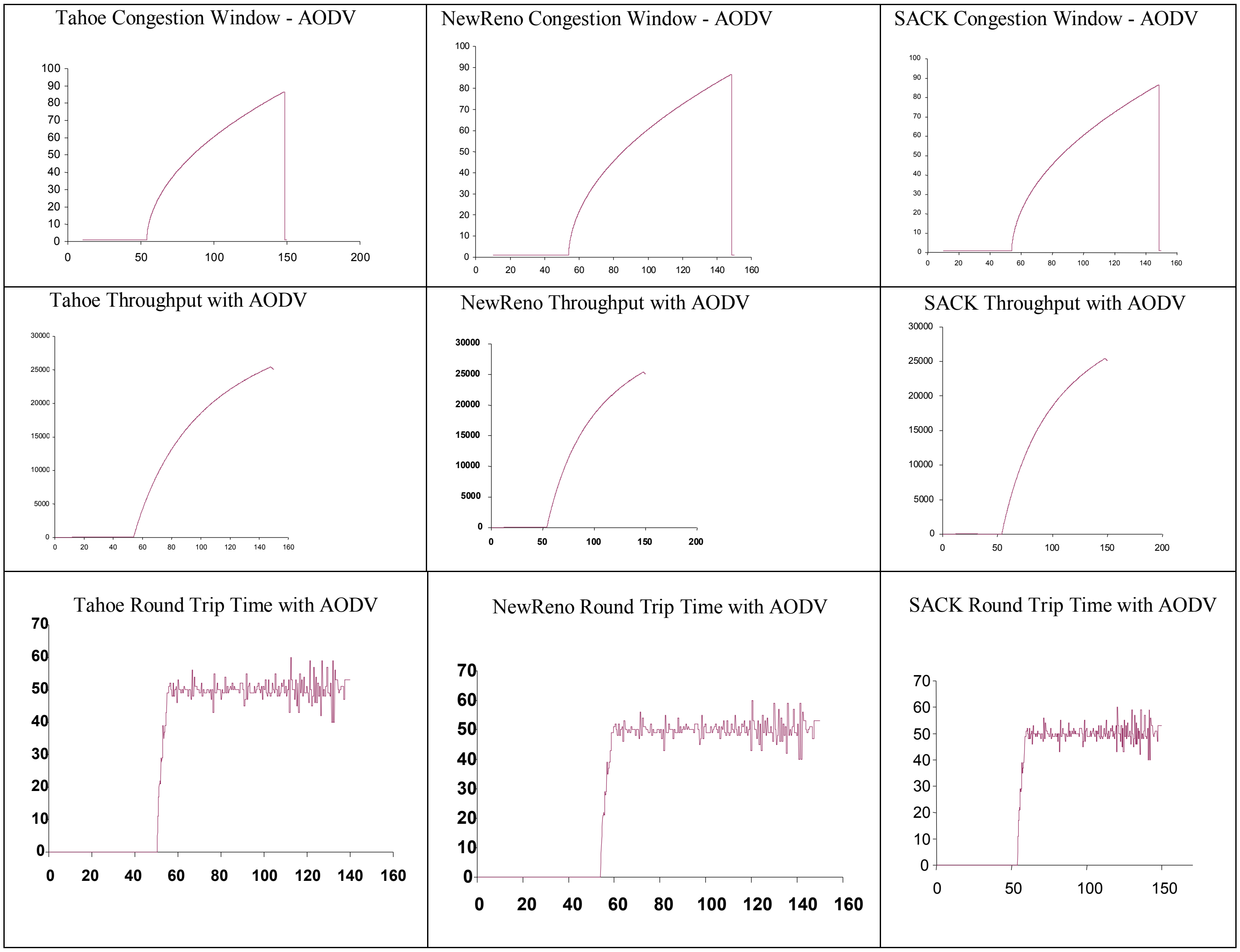

Figure 4. AODV results with Tahoe, NewReno, and SACK.

8. Conclusion

Several factors can affect the underlying TCP performance. Routing protocols are important for finding a route to the destination for information packet delivery. Routing protocols are designed in such away to find the route accurately and quickly. In Mobile Ad-hoc Networks, nodes are always mobile therefore route-changing possibilities are dominant. Routing protocols are their best in finding the optimised route to make the communication possible. Is routing protocols working mechanisms have any effect on the underlying TCP performance? What is the effect of background traffic on TCP performance?

This paper has presented a performance comparison of the DSDV, DSR and AODV routing protocols when combined with varying TCP algorithms along with background traffic. After briefly studying and discussion, it has been found that routing protocols have effects on the underlying TCP performance. In case of 260kbps background traffic no routing protocol produced more than 0.66% of throughput except DSR. DSR when evaluated with SACK TCP gave extremely good results almost ten folds more than the others i.e. 6.66% throughput. AODV is consistent with all of the TCP algorithms tested. Similarly, AODV is found very slow in connection establishment as compared to DSDV and DSR and therefore very low amount of data is transmitted .i.e. the amount of transmitted bytes is less than DSR and DSDV. The same combination when was evaluated without background traffic. It was found that 16.26 % drop was observer in the total transmitted bytes.

Thus none of the MANETs routing protocol is found of providing optimised performance with all of the TCP algorithms. A good routing protocol should be one having consistent and good performance with all of the TCP algorithms as different operating systems comes with implementation of different underlying TCP implementation. Anyway by comparison, it is found that DSR produces good performance with all of the TCP algorithms as compared to DSDV and AODV but AODV is consistent with all of the TCP algorithms. Thus it can be concluded that source initiated on-demand routing protocols can produce good results with TCP algorithms, even in the case of background traffic, although further study of additional source initiated on-demand routing protocols is needed to validate this result.

Corresponding Author:
Salman Faiz Solheria
Department of Computer Science
Sarhad University of Science & Technology
KPK, Pakistan
E-mail: salman@suit.edu.pk

References

1. Bakht, H. A Study of Routing Protocols for Mobile Ad-hoc Networks. In proceedings of the 1st International Computer Engineering Conference 2004, Cairo , Egypt, 2010.
2. Broch J, Maltz DA, Johnson DB, Hu YC, Jetcheva J. A performance comparison of multi-hop wireless ad hoc network routing protocols. Paper presented at the Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking 1998, Dellas, Texas, USA, 1998; 85-97.
3. Clark DD, Hoe JC. Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes. Tchnical Report, Massachusetts Institute of Technology 1995.
4. Comer DE, Stevens DL. Internetworking with TCP/IP (vol. 2, 2nd ed.): design, implementation, and internals. Prentice-Hall Inc. 1994; 612.
5. Das SR, Casta R, Yan J. Simulation-based performance evaluation of routing protocols for mobile ad hoc networks. Mob. Netw. Appl. 2000; 5(3):179-189.
6. Dyer TD, Boppana RV. A comparison of TCP performance over three routing protocols for mobile ad hoc networks. In proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing 2001, Long Beach, CA, USA; 56-66.
7. Fall K, Floyd S. Simulation-based comparisons of Tahoe, Reno and SACK TCP. SIGCOMM Comput. Commun. Rev. 1996; 26(3): 5-21.
8. Hoe JC. Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes. Master Thesis, Master of Science in Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Massachusetts, USA, 1995;69.
9. Jacobson V. Congestion avoidance and control. SIGCOMM Comput. Commun. Rev. 1995; 25(1):157-187.
10. Johansson P, Larsson T, Hedman N, Mielczarek B, Degermark, M. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking 1999, Seattle, Washington, USA, 1999;195-206.
11. Johnson DB, Maltz DA. Dynamic Source Routing in Ad Hoc Wireless Networks. Mobile Computing 1996; 153-181.
12. Johnson DB, Maltz DA. Truly seamless wireless and mobile host networking. Protocols for adaptive wireless and mobile networking. IEEE Personal Communications 1996; 3(1): 34-42.

13. Mathis M, Mahdavi J. Forward acknowledgement: refining TCP congestion control. SIGCOMM Comput. Commun. Rev. 1996; 26(4): 281-291.
14. Perkins CE, Bhagwat P. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. SIGCOMM Comput. Commun. Rev. 1994; 24(4): 234-244.
15. Perkins CE, Royer EM. Ad-hoc On-Demand Distance Vector Routing. In proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications 1999;90.
16. Pucha H, Das SM, Hu YC. The performance impact of traffic patterns on routing protocols in mobile ad hoc networks. In proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems 2004, Venice, Italy, 2004;211-219.
17. Steenstrup M. (Ed.). Routing in communications networks: Prentice Hall International (UK) Ltd, 1995; 399.
18. Trung HD, Benjapolakul W, Duc PM. Performance evaluation and comparison of different ad hoc routing protocols. Comput. Commun. 2007; 30(11-12): 2478-2496.

2/15/2021