Comparison of Drop Rates in Different TCP Variants against Various Routing Protocols

A. R. Britto Pradeep
PG Scholar
IT Department
Karunya University, India

N. Dhinakaran
Assistant Professor
IT Department
Karunya University, India

P. Anitha Christy Angelin
Assistant Professor
IT Department
Karunya University, India

ABSTRACT
In a network, the most common transport protocol is the Transmission Control Protocol. The Transmission Control Protocol comes in many variants like TCP, Tahoe, Reno, NewReno, Vegas, STCP and so on. Each of these variants would work differently in different networks according to the parameters of that network. On the other hand, there are mainly four common routing protocols used in networks like DSDV, DSR, AODV and TORA. In this paper, we have simulated different networks with differing parameters to analyze the behavior of the most common protocols DSDV and AODV with different variants of TCP. By creating different networks in ns2 simulator, we could deeply analyze the behavior of the protocols with these TCP variants in the basis of the amount of packet drops in each case. The lesser the amount of drops the better the algorithm. This paper implicitly analyses which TCP variant has lesser drop rates with which routing protocol.

General Terms
Routing Protocols, TCP Variants, Packet Drops.

Keywords
Transmission Control Protocol, TCP Variants, DSDV, AODV.

1. INTRODUCTION
The most common transport protocol used in internet for data transmission is Transmission Control Protocol. There are many variants of TCP each variant being used for a specific purpose. The five variants we use in this paper are TCP, Tahoe, Reno, NewReno Vegas. We call these variants as Agents. First, we take each algorithm and compare the variants for 50 nodes. Then, the number of nodes is increased to 100. In each case, the behavior of the TCP variants is observed in terms of packet drops. The remaining part of this segment will explain briefly the five TCP agents and the two protocols DSDV and AODV which are being used in this paper.

2. ROUTING PROTOCOLS

2.1 DSDV
In the Destination Sequenced Distance Vector protocol routing messages are exchanged between neighbouring mobile nodes (i.e., mobile nodes that are within range of one another). Routing updates may be triggered or routine. Updates are triggered in case routing information from one of the neighbours forces a change in the routing table. A packet for which the route to its destination is not known is cached while routing queries are sent out. The packets are cached until route-replies are received from the destination. There is a maximum buffer size for caching the packets waiting for routing information beyond which packets are dropped. All packets destined for the mobile node are routed directly by the address dmux to its port dmux. The port dmux hands the packets to the respective destination agents. A port number of 255 is used to attach routing agent in mobile nodes. The mobile nodes also use a default-target in their classifier (or address demux). In the event a target is not found for the destination in the classifier (which happens when the destination of the packet is not the mobile node itself), the packets are handed to the default-target which is the routing agent. The routing agent assigns the next hop for the packet and sends it down to the link layer. The routing protocol is mainly implemented in C++.

The Route Selection Strategy of DSDV is such that a router uses only the recently updated information. In case of repetitive sequence numbers, it goes for the best metric. Stale entries are deleted regularly. The advantages of DSDV being, it can be used for creating ad hoc networks with small number of nodes. Also, DSDV promises loop free path. The disadvantage is regular updating of routing tables, causes battery depletion and bandwidth exhaustion. The same reason states for its non-usability with highly dynamic networks as in [1].

2.2 AODV
The Ad hoc On-demand Distance Vector protocol is a combination of both DSR (Dynamic Source Routing) and DSDV protocols. It has the basic route-discovery and route-maintenance of DSR and uses the hop-by-hop routing, sequence numbers and beacons of DSDV. The node that wants to know a route to a given destination generates a ROUTE REQUEST. The route request is forwarded by intermediate nodes, which also creates a reverse route for itself from the destination. When the request reaches a node with route to destination it generates a ROUTE REPLY containing the number of hops required to reach the destination. All nodes that participate in forwarding this reply to the source node create a forward route to destination. This state created from each node from source to destination is a hop-by-hop state and not the entire route as is done in source routing. The advantages of AODV are that the routes are established only on demand and the sequence numbers are used only to find the latest route to the destination as in [2]. It also has a reduced connection setup delay. The disadvantages are that stale information, heavy overheads caused by multiple ROUTE REPLY messages and unnecessary bandwidth consumption caused by periodic beaconing.

3. TCP AND ITS VARIANTS

3.1 TCP
TCP provides a connection oriented, reliable, byte stream service. The term connection-oriented means the two applications using TCP must establish a TCP connection with each other before they can exchange data as shown in the figure

1. It is a full duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction. TCP includes a flow-control mechanism for each of these byte streams that allow the receiver to limit how much data the sender can transmit. TCP also implements a congestion-control mechanism. TCP is a reliable connection oriented end-to-end protocol which has many mechanisms to provide reliable communication. But a small number of packets are lost due to congestion and buffer overflow. In such cases, TCP ensures reliability by using sequence numbers and time-out intervals. The packet of the particular sequence number is resent after the time-out timer runs out. TCP runs on the concept of "Conservation of Packets" [3]. The TCP provides different facilities as discussed below in the following list.

i. Stream Data Transfer:

TCP transfers a contiguous stream of bytes. TCP does this by grouping the bytes in TCP segments, which are passed to IP for transmission to the destination. TCP decides how to segment the data and forwards the data at its own convenience.

ii. Reliability:

TCP assigns a sequence number to each byte transmitted, and expects a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. The receiving TCP uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.

iii. Flow Control:

The receiving TCP, when sending an ACK back to the sender, also indicates to the sender the number of bytes it can receive beyond the last received TCP segment, without causing overrun and overflow in its internal buffers. This is sent in the ACK in the form of the highest sequence number it can receive without problems.

iv. Multiplexing:

To allow for many processes within a single host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer this forms a socket. A pair of sockets uniquely identifies each connection.

v. Logical Connections:

The reliability and flow control mechanisms described above require that TCP initializes and maintains certain status information for each data stream. The combination of this status, including sockets, sequence numbers and window sizes, is called a logical connection. Each connection is uniquely identified by the pair of sockets used by the sending and receiving processes.

vi. Full Duplex:

TCP provides for concurrent data streams in both directions.

3.2 TCP Tahoe

TCP Tahoe is the congestion control mechanism suggested by Van Jacobson. The actual TCP data transmission is clocked by the acknowledgements received. But at the start of the transmission, there would not be any acknowledgement. To overcome this, the Tahoe suggests a mechanism called "slow-start". According to this mechanism, the congestion window size is taken as 1 at the beginning of start or a restart of data transmission. After sufficient acknowledgements are received, the congestion window size is additionally increased. After congestion is achieved, the window size is multiplicatively decreased. This is called Additive Increase Multiplicative Decrease [4]. Whenever a packet is lost, the "go back n" method is used, and the entire pipe is emptied. This results in a high-bandwidth delay.

3.3 TCP Reno

TCP Reno has all the advantages of Tahoe like the slow start mechanism and the time-out intervals. Also, it has some intelligent mechanisms to detect the packet losses previously. After each packet loss, the entire pipe is not emptied. It uses a Fast Retransmit mechanism in which when 3 duplicate acknowledgements are received, it is understood that there is packet loss. Hence even before the actual packet loss is detected, the packet is retransmitted. It has the disadvantage of reducing the window size more than required and hence cannot afford Fast Recovery [5]. If window size is reduced very much, then the normal course grained timeout.

3.4 TCP New Reno

The TCP New Reno [6] is more advanced than TCP Reno. It is able to detect multiple packet losses. It also enters the fast recovery mechanism like Reno, but it does not end up reducing the congestion window size. It waits till the acknowledgements of all the congested packets are received. The actual disadvantage of the New Reno is that it takes a whole Round Trip Time to detect a single packet loss.

3.5 TCP Vegas

The TCP Vegas [7] is a modified version of Reno. It works on the proactive measures to control congestion rather than reactive measures. It uses an algorithm to check for timeouts. It also overcomes the problem of requiring enough duplicate acknowledgements to detect a packet loss. It also uses a slightly modified slow start mechanism. It has the mechanism to detect congestion even before packet losses occur, but it also retains the other mechanisms of Reno and Tahoe. Overall, the Vegas has a new retransmission mechanism, a modified slow start algorithm and congestion avoidance scheme.

4. TCP CHARACTERSTICS

4.1 Congestion Window

In any network, when the number of packets in the network increases randomly, it leads to message delay or at times loss of data. This situation is called network congestion. In TCP, to avoid congestion, we use congestion windows, which determine the number of byes that can be outstanding at any time. This is a means of stopping the link between two places from getting overloaded with too much traffic. The size of the window is calculated by estimating how much congestion is there between the two places. Usually, it is the sender that maintains the congestion window. When a connection is set up, the congestion window is set to the Maximum Segment Size allowed on that connection. Further variance in the collision window is dictated by an Additive Increase/Multiplicative Decrease approach. This

means that if all segments are received and the acknowledgements reach sender on time, some constant is added to the window size. The window keeping growing linearly until a timeout occurs or the receiver reaches its limit. If a timeout occurs, the window size is halved [8].

4.2 Congestion Control

TCP uses a number of mechanisms to achieve high performance and avoid 'congestion collapse', where network performance can degrade by several orders of magnitude [9]. These mechanisms control the rate of data entering the network, keeping the data flow below a rate that would trigger collapse. Acknowledgements for data sent are used by senders to infer network conditions between TCP sender and receiver. Modern implementations of TCP contain four intertwined algorithms: slow-start, congestion avoidance, fast retransmit and fast recovery.

4.2.1 Slow Start

It operates by observing that the rate at which new packets should be injected into the network is the rate at which the acknowledgments are returned by the other end. Slow start adds another window to the sender's TCP: the congestion window, called "cwnd". When a new connection is established with a host on another network, the congestion window is initialized to one segment (i.e., the segment size announced by the other end, or the default, typically 536 or 512). Each time an ACK is received, the congestion window is increased by one segment. The sender can transmit up to the minimum of the congestion window and the advertised window.

The congestion window is flow control imposed by the sender, while the advertised window is flow control imposed by the receiver. The former is based on the sender's assessment of perceived network congestion; the latter is related to the amount of available buffer space at the receiver for this connection. The sender starts by transmitting one segment and waiting for its ACK. When that ACK is received, the congestion window is incremented from one to two, and two segments can be sent. When each of those two segments is acknowledged, the congestion window is increased to four. This provides an exponential growth, although it is not exactly exponential because the receiver may delay its ACKs, typically sending one ACK for every two segments that it receives. At some point the capacity of the internet can be reached, and an intermediate router will start discarding packets. This tells the sender that its congestion window has gotten too large. Early implementations performed slow start only if the other end was on a different network. Current implementations always perform slow start.

4.2.2 Congestion Avoidance

Congestion can occur when data arrives on a big pipe (a fast LAN) and gets sent out a smaller pipe (a slower WAN). Congestion can also occur when multiple input streams arrive at a router whose output capacity is less than the sum of the inputs. Congestion avoidance is a way to deal with lost packets. The assumption of the algorithm is that packet loss caused by damage is very small (much less than 1%), therefore the loss of a packet signals congestion somewhere in the network between the source and destination.

There are two indications of packet loss: a timeout occurring and the receipt of duplicate ACKs. Congestion avoidance and slow start are independent algorithms with different objectives. But when congestion occurs TCP must slow down its transmission rate of packets into the network, and then invoke slow start to get things going again. In practice they are implemented together. Congestion avoidance and slow start require that two variables be maintained for each connection: a congestion window, cwnd, and a slow start threshold size, ssthresh. The combined algorithm operates as follows:

1. Initialization for a given connection sets cwnd to one segment and ssthresh to 65535 bytes.

2. The TCP output routine never sends more than the minimum of cwnd and the receiver's advertised window.

3. When congestion occurs (indicated by a timeout or the reception of duplicate ACKs), one-half of the current window size (the minimum of cwnd and the receiver's advertised window, but at least two segments) is saved in ssthresh. Additionally, if the congestion is indicated by a timeout, cwnd is set to one segment (i.e., slow start).

4. When new data is acknowledged by the other end, increase cwnd, but the way it increases depends on whether TCP is performing slow start or congestion avoidance. If cwnd is less than or equal to ssthresh, TCP is in slow start; otherwise TCP is performing congestion avoidance. Slow start continues until TCP is halfway to where it was when congestion occurred (since it recorded half of the window size that caused the problem in step 2, and then congestion avoidance takes over. Slow start has cwnd begin at one segment, and be incremented by one segment every time an ACK is received.

As mentioned earlier, this opens the window exponentially: send one segment, then two, then four, and so on. Congestion avoidance dictates that cwnd be incremented by segsize*segsize/cwnd each time an ACK is received, where segsize is the segment size and cwnd is maintained in bytes. This is a linear growth of cwnd, compared to slow start's exponential growth. The increase in cwnd should be at most one segment each round-trip time (regardless how many ACKs are received in that RTT), whereas slow start increments cwnd by the number of ACKs received in a round-trip time.

4.2.3 Fast Retransmit

TCP may generate an immediate acknowledgment (a duplicate ACK) when an out- of-order segment is received. This duplicate ACK should not be delayed. The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected. Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received. It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire.

4.2.4 Fast Recovery

After fast retransmit sends what appears to be the missing segment, congestion avoidance, but not slow start is performed. This is the fast recovery algorithm. It is an improvement that allows high throughput under moderate congestion, especially for large windows. The reason for not performing slow start in this case is that the receipt of the duplicate ACKs tells TCP more than just a packet has been lost. Since the receiver can only generate the duplicate ACK when another segment is received, that segment has left the network and is in the receiver's buffer. That is, there is still data flowing between the two ends, and TCP does not want to reduce the flow abruptly by going into slow start. The fast retransmit and fast recovery algorithms are usually implemented together as follows.

1. When the third duplicate ACK in a row is received, set ssthresh to one-half the current congestion window, cwnd, but no less than two segments. Retransmit the missing segment. Set cwnd to ssthresh plus 3 times the segment size. This inflates the congestion window by the number of segments that have left the network and which the other end has cached.

2. Each time another duplicate ACK arrives, increment cwnd by the segment size. This inflates the congestion window for the additional segment that has left the network. Transmit a packet, if allowed by the new value of cwnd.

3. When the next ACK arrives that acknowledges new data, set cwnd to ssthresh (the value set in step 1. This ACK should be the acknowledgment of the retransmission from step 1, one round-trip time after the retransmission. Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK. This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost.

4.3 Flow Control

In computer networking, flow control is the process of managing the data rate between two nodes to prevent a fast sender from outrunning a slow receiver. It provides mechanism for the receiver to control the transmission speed, so that it is not overwhelmed. Flow Control should be distinguished from congestion control, which is used for controlling the flow of data when congestion has occurred actually.In a connection between a client and a server, the client tells the server the number of bytes it is willing to receive at one time from the server; this is the client's receive window, which becomes the server's send window. Likewise, the server tells the client how many bytes of data it is willing to take from the client at one time; this is the server's receive window and the client's send window. Since the window size can be used in this manner to manage the rate at which data flows between the devices at the ends of the connection, it is the method by which TCP implements flow control, one of the “classical” jobs of the transport layer. Flow control is vitally important to TCP, as it is the method by which devices communicate their status to each other.

By reducing or increasing window size, the server and client each ensure that the other device sends data just as fast as the recipient can deal with it. Flow control is a technique whose primary purpose is to properly match the transmission rate of sender to that of the receiver and the network. It is important for the transmission to be at a high enough rates to ensure good performance, but also to protect against overwhelming the network or receiving host. Congestion control is primarily concerned with a sustained overload of network intermediate devices such as IP routers. TCP uses the window field, briefly described previously, as the primary means for flow control. During the data transfer phase, the window field is used to adjust the rate of flow of the byte stream between communicating TCPs.

Flow control mechanisms can be classified by whether or not the receiving node sends some feedback to the sender. It is important because it is possible for a sender to send data at a faster rate than the receiver can receive and process them.

4.4 Sliding Window

The sliding window is a technique used in tcp to provide flow control such that all packets arrive in the same sequential order in which they were sent. Sliding Window Protocols are a feature of packet-based data transmission protocols. They are used anywhere reliable in-order delivery of packets is required, such as in the data link layer (OSI model) as well as in TCP (transport layer of the OSI model). Conceptually, each portion of the transmission (packets in most data link layers, but bytes in TCP) is assigned a unique consecutive sequence number, and the receiver uses the numbers to place received packets in the correct order, discarding duplicate packets and identifying missing ones. The problem with this is that there is no limit of the size of the sequence numbers that can be required. By placing limits on the number of packets that can be transmitted or received at any given time, a sliding window protocol allows an unlimited number of packets to be communicated using fixed-size sequence numbers.

For the highest possible throughput, it is important that the transmitter is not forced to stop sending by the sliding window protocol earlier than one round-trip delay time (RTT). The limit on the amount of data that it can send before stopping to wait for an acknowledgment should be larger than the bandwidth-delay product of the communications link. If it is not, the protocol will limit the effective bandwidth of the link.

5. Evaluation Environment

The experiments were conducted in two different scenarios and in ten different ways. In all cases, some parameters were made constant. The parameters that were changed were the transmission agent and the routing protocol. The ns2 simulator [10] was used to analyze the protocols against the TCP variants. The table 1 shows the parameters that were common in all the scenarios.

Table 1. Common Properties in Scenarios

Characteristics	Value
Channel Type	*Wireless Channel*
Radio Propagation	*Two Ray Ground*

Network Interface	*Wireless Physical Interface*
MAC Type	*802.11*
Interface Queue	*Drop Tail*
Antenna	*Omni Antenna*
Maximum Packets in Interface Queue	*50*
Simulation Time	*200s*

The TCP agent was changed for each scenario as TCP, TCP Tahoe, TCP Reno, TCP NewReno and TCP Vegas with two routing protocols DSDV and AODV. Also, each experiment was conducted for 50 mobile nodes and 100 mobile nodes. Hence, totally 20 different experiments were conducted from which four graphs were obtained for the observation of the performance of different TCP variants for DSDV and AODV routing protocols.

6. RESULTS AND ANALYSIS

6.1 AODV in 50 Nodes

In this analysis, five different experiments were conducted. In each experiment, a different TCP variant was used. Each time the routing protocol was AODV. There was no change in the experimental environment which is described in the above table 1. The total number of nodes was 50. The nodes did not have any mobility. All nodes were kept stable for the full course of the evaluation. In the graph, the variants are denoted in short as T for TCP Tahoe, R for TCP Reno, NR for TCP New Reno and V for TCP Vegas. The Fig 1 shows that for the same scenario and parameters, the performance of TCP Tahoe seems to be convincing because it has very little amount of drops compared to the other variants of TCP. This is in the case of 50 nodes which use the AODV protocol for routing. The second best next to TCP Tahoe is TCP Vegas. The performance of the other agents can be in the order of TCP, TCP Reno, TCP NewReno and TCP itself.

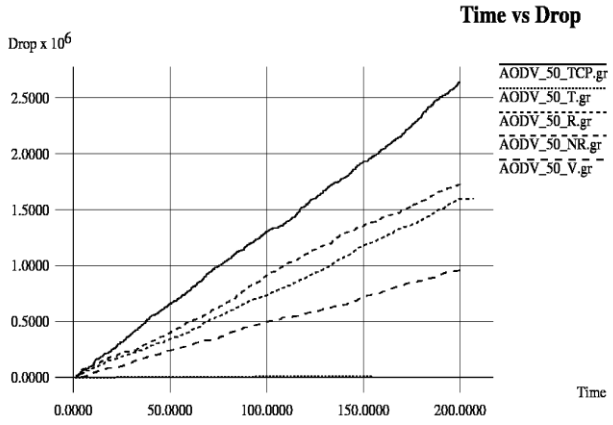

Fig 1: TCP Drop Rates with AODV in 50 Nodes

6.2 AODV in 100 Nodes

The Fig 2 shows the graph between time and the number of dropped packets for each scenario using different TCP agent. At the start of the experiment, the TCP Vegas has a very little number of drops per time.

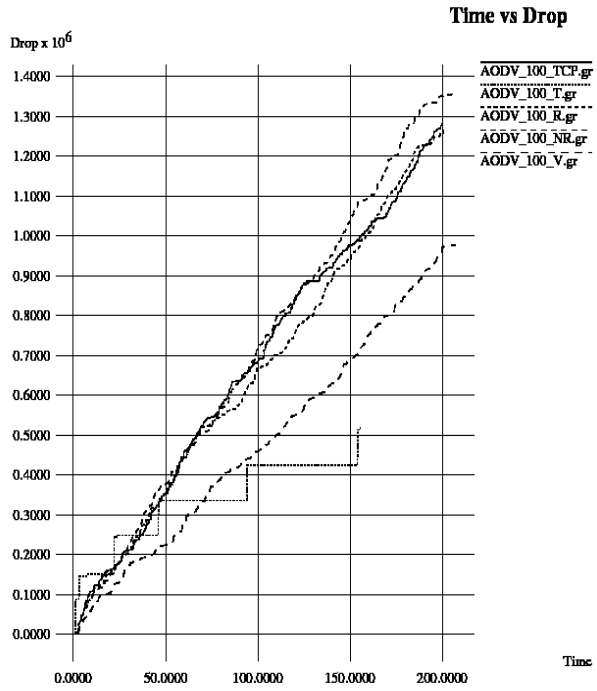

Fig 2: TCP Drop Rates with AODV in 100 Nodes

But as time goes on, the scene changes with the amount of packet drops in Vegas slowly starts increasing. The other variants like TCP, TCP Reno and TCP NewReno start with the same amount of packet drops at the start of the simulation. But on the course of the simulation, these three variants have a rapid change in the amount of packet drops. It constantly keeps on varying from high to low and vice versa.

The TCP Tahoe variant starts with a very high drop rate. But as the time increases on to the 2/4th of the simulation, the TCP Tahoe shows a very good sign of reduction in packet drop rate. And along with time, the drop rate slowly keeps rising. But at the end of the simulation, the TCP Tahoe stands at the bottom of the graph. It denotes that even when the number of nodes is increased to double the previous number, the performance of TCP Tahoe is relatively better than the other variants of TCP.

6.3 DSDV in 50 Nodes

The Fig 3 shows the graph of Time versus the Number of Dropped Packets, in the scenario where the TCP agent is changed each time and the number of nodes is 50, with DSDV as the routing protocol. The TCP NewReno has the highest number of packet drops versus time, according to the shown graph. The TCP and TCP Reno are moving along the same line in the amount of packet drops versus time. They are the second

largest packet dropping agents next to TCP NewReno. The next one is the TCP Vegas which starts with lesser packet drops and as time increases, the number of packet drops also increases. But it is lesser than the other variants like TCP, TCP reno and TCP NewReno. In DSDV also, similar to AODV, the TCP Tahoe variant is the agent that has least amount of packet drops against time. In case of DSDV also, the amount of packet drops in Tahoe variant is highly negligible.

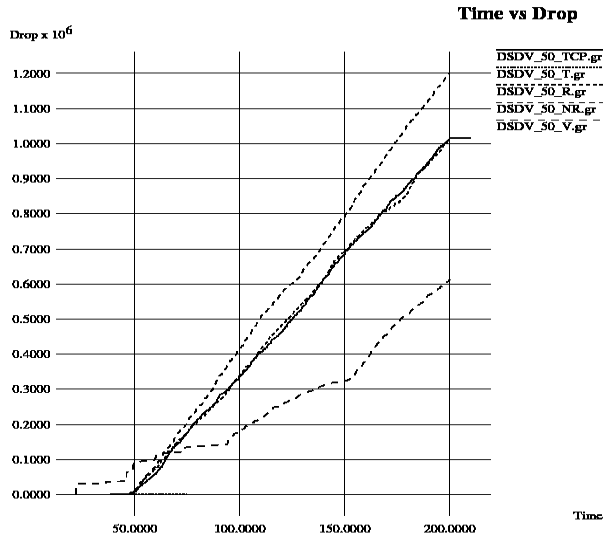

Fig 3: TCP Drop Rates with DSDV in 50 Nodes

6.4 DSDV in 100 Nodes

The Fig 4 shows the graph between the Time and the number of packet drops, by different TCP agents when using DSDV as the routing protocol with 100 nodes. The graph clearly states that the TCP NewReno is the largest packet dropping agent when DSDV is used with 100 mobile modes. The next highest dropping agent is the TCP itself. The TCP Vegas is the third highest packet dropping agent next to the TCP NewReno and the TCP itself. The TCP Vegas agent started with a lesser number of packet drops as in the previous scenario. But as time increased, the number of packets being dropped increased in the case of TCP Vegas. The next agent is TCP Reno which has closer curve to the TCP Vegas. But at the end of the simulation, the TCP Reno has lesser number of packets dropped when compared to the TCP Vegas variant. As in all the previous cases, the TCP Tahoe has the least amount of packet drops when compared to all the other TCP Variants. Even when increasing the number of nodes to 100, when using DSDV as the routing protocol, the TCP Tahoe proves to be the efficient agent according to the number of packets dropped.

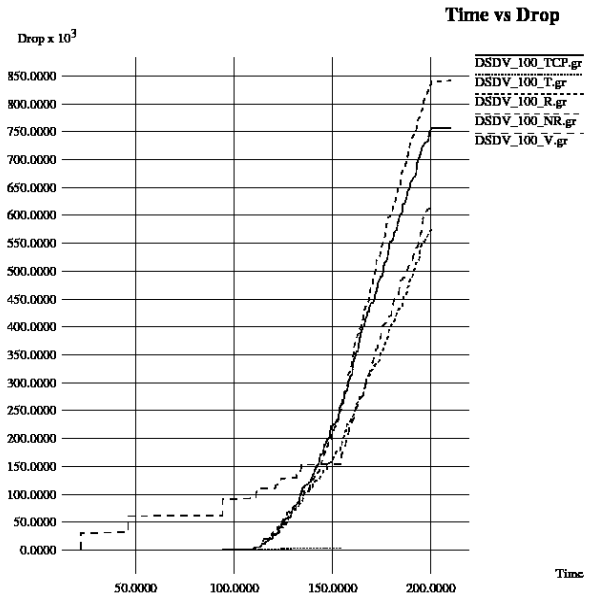

Fig 4: TCP Drop Rates with DSDV in 100 Nodes

7. CONCLUSION

To compare the performances of different TCP variants like TCP, TCP Reno, TCP NewReno, TCP Vegas and TCP Tahoe with the routing protocols DSDV and AODV, we have experimented in 20 different ways to find that TCP Tahoe has the least number of packet drops against the simulation time. Some of the other variants, though they start with a lesser number of packet drops, the TCP Tahoe variant has always the least amount of packet drops in all cases like when using AODV and DSDV, be it 50 nodes or 100 nodes. Hence, irrespective of the number of nodes being increased and the simulation time being increasing, the TCP Tahoe always has the least packet drops. In future, the same experiments can be repeated with increased number of nodes and with other routing protocols. Also, these 20 experiments were done assuming that all the nodes were in constant position. Hence, in future, the efficiency of the TCP agents can be studied after introducing some amount of mobility to the nodes.

8. REFERENCES

[1] The Working Concept of DSDV Protocol, [Online] http:// .wikipedia.org/Wiki/Destiation-Sequenced_Distance_Vector_routing

[2] The Working Concept of AODV Protocol, [Online] http:// .wikipedia.org/Wiki/Ad_hoc_On-Demand_Distance_Vector_Routing.

[3] W. Richard Stevens. TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994

[4] K. Fall and S. Floyd, "Simulation-based comparison of Tahoe, Reno, and SACK TCP," Computer Communication Review, vol. 26, pp. 5--21, July 1996.

[5] Jitendra Padhye , Victor Firoiu , Donald F. Towsley , James F. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," IEEE/ACM Transactions on Networking (TON), v.8 n.2, p.133-145, April 2000

[6] S.Floyd, T.Henderson "The New-Reno Modification to TCP's Fast Recovery Algorithm" RFC 2582, Apr 1999.

[7] Lawrence S. Brakmo and Larry L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," IEEE Journal on Selected Areas in Communications, vol.13, pp.1465-1480, October 1995.

[8] V. Jacobson, "Congestion Avoidance and Control", SIGCOMM '88, Sept. 1988, pp. 314-329.

[9] S.Floyd, "Promoting the Use of End-toEnd Congestion Control in the Internet", IEEE/ACM Transactions on Networking, vol. 7, no.4, August 1999

[10] NS-2 homepage, [Online] http://www.isi.edu/nsnam/ns, 2011