STUDY OF TCP VARIANTS OVER WIRELESS NETWORK

'DEVEN DRA SINGH KUSHWAHA, 'VIKASH K SINGH, 'SHAIBYA SINGH, 'SONAL SHARMA

1,2,3,4 Assistant Professor, Dept. of Computer Science, Indira Gandhi National Tribal University, Amarkantak
E-mail: devendra2904@gmail.com, drvksingh76@gmail.com, shaibyaigntu@gmail.com, Sonal_sdl@yahoo.co.in

Abstract: The performance of wireless ad-hoc networks is affected extensively by TCP; the magnitudes of lost packets are usually high under TCP due to congestion. In a mobile ad hoc network, temporary link failures and route changes occur frequently, with the assumption that all packet losses are due to congestion, TCP performs poorly in such an environment. TCP was designed specifically for wired, reliable network thus, any packet loss is attributed to congestion in the network. Transport Control Protocol (TCP), a basic communication language, consists of a set of rules that control communication. There are many improved versions of TCP, which were designed time to time as per necessities. This paper reveals the study of different TCP variants specifically Tahoe, Reno, New Reno, Selective Acknowledgment (SACK) and TCP Vegas. TCP New Reno algorithm and TCP Vegas are explained with new structure mechanism and new congestion avoidance and modified slow start mechanisms. This paper also presents a comparative analysis of TCP variants on the basis of algorithm.

Keywords: Congestion Avoidance, Tahoe, Reno, New Reno Fast Recovery, Fast Retransmission and Slow Start, Mobile Ad-hoc Network

I. INTRODUCTION

Mobile ad hoc networks (MANETs) are collections of mobile nodes, dynamically forming a temporary network without centralized administration. These nodes can be arbitrarily located and are free to move randomly at any given time, thus allowing network topology and interconnections between nodes to change rapidly and unpredictably. There has been significant research activity over the past 10 year into performance of such networks with the view to develop more efficient and robust TCP variants. Transmission control protocol (TCP) provides reliability, end-to-end congestion control mechanism, byte stream transport mechanism, flow control, and congestion control. Comparing to wire networks, there are many different characteristics in wireless environments, which makes TCP congestion control mechanism is not directly suitable for wireless networks and many improved TCP congestion control mechanisms have been presented. However, TCP in its present form is not well suited for mobile ad hoc networks. In addition to all links being wireless, frequent route failures due to mobility can cause serious problems to TCP as well. Route failures can cause packet drops at the intermediate nodes, which will be misinterpreted as congestion loss.

TCP (Transmission Control Protocol) is widely used by many Internet services including HTTP (and World Wide Web) and FTP (File Transfer Protocol). Even if the network infrastructure may change in the future, it is very likely that TCP and its applications would be continuously used. However, TCP Tahoe and Reno versions (and their variants), which are widely used in the current Internet, are not perfect in terms of throughput and fairness among connections, as having been shown in the past literatures. Therefore, active researches on TCP have been made, and many improvement mechanisms have been proposed. Among them, a TCP Vegas version is one of the most promising mechanisms by its high performance. TCP Vegas enhances the congestion avoidance algorithm of TCP Reno. In essence, TCP Vegas dynamically increases/decreases its sending window size according to observed RTT (Round Trip Times) of sending packets, whereas TCP Tahoe/Reno only continues increasing its window size until packet loss is detected.

The author in concludes through simulation and implementation experiments that TCP Vegas can obtain even 40% higher throughput than TCP Reno. However, we need to consider a migration path when anew protocol is deployed in the operating network, i.e., the Internet. It is important to investigate the effect of existing TCP versions (Tahoe and Reno) on TCP Vegas in the situation where those different versions of TCP co-exist in the network.

The authors in have pointed out that when connections of TCP Reno and Vegas share the bottleneck link, the Vegas connection may suffer from significant unfairness. However, the authors have assumed that only a single TCP Reno connection shares the link with another TCP Vegas connection.

Most of research paper are concentrating on congestion, corruption control and improve retransmission time out condition, improving performance metrics and security threats of protocol. Hence this thrust area of mobile ad-hoc network become the choice of interest for us.

This paper is organized in such a manner that section II describes Congestion Control Algorithm followed by performance evaluation of TCP variants and
comparison studies. Section III describes corruption control by TCP. Section IV is based on study of different TCP variants, and Section V presents a comparative study of various TCP variants. The conclusion is covered by section VI.

II. CONGESTION CONTROL ALGORITHM

The major responsibilities of TCP include congestion control, flow control, in-order delivery of packets, and reliable transportation of packets. Congestion control deals with excess traffic in the network which may lead to degradation in the performance of the network, whereas flow control controls the per-flow traffic such that the receiver capacity is not exceed. The TCP sender starts the session with a congestion window value of one MSS. It sends out one MSS and waits for the retransmission ACK. Once the ACK is received within the retransmission timeout (RTO) period, the congestion window is doubled and two MSSs are originated. This doubling of the congestion window with every successful acknowledgment of all the segments in the current congestion window is called slow-start or exponential start show in figure 1 and it continues until the congestion window reaches the slow-start threshold.

Once it reaches the slow-start threshold, it grows linearly, adding one MSS to the congestion window on every ACK received. This linear growth, which continues until the congestion window reaches the receiver window, is called congestion avoidance, show in figure 1 as it tries to avoid increasing the congestion window exponentially, which will surely worsen the congestion in the network. TCP updates the RTO period with the current round-trip delay calculated on the arrival of every ACK packet. If the ACK packet does not arrive within the RTO period, then it assumes that the packet is lost. TCP assumes that the packet loss is due to the congestion in the network and it invokes the congestion control mechanism.

The TCP sender does the following during congestion control:
(i) Reduces the slow-start threshold to half the current congestion window or two MSSs whichever is larger.
(ii) Resets the congestion window size to one MSS,
(iii) Activates the slow-start algorithm, and
(iv) Resets the RTO with an exponential back-off value which doubles with every subsequent retransmission.

The TCP sender also assumes a packet loss if it receives three consecutive duplicate ACKs (DUPACKs). Upon reception of three DUPACKs, the TCP sender retransmits the oldest unacknowledged segment. This is called the fast retransmit scheme show in figure 1. Among the several extensions of TCP, some of the important schemes are discussed below. The regular TCP which was discussed above is also called as TCP Tahoe.

TCP Reno is similar to TCP Tahoe with fast recovery. On timeout or arrival of three DUPACKs, the TCP Reno sender enters the fast recovery and congestion window size to half the size of the current congestion window, and increments the congestion window linearly with every subsequent DUPACK. On reception of a new the TCP Reno resets the congestion window with the slow-start threshold and enters the congestion avoidance phase similar to TCP Tahoe.

TCP with selective ACK (SACK) improves the performance of TCP by using the selective ACKs provided by the receiver. The receiver sends a SACK instead of an ACK, which contains a set of SACK blocks. These SACK blocks contain information about the recently received packets which is used by the TCP sender while retransmitting the lost packets.

III. CORRUPTION CONTROL

In wireless mobile networks, usually, loss of packets is because of corruption of data on the wireless link. Wireless networks are characterized by large error rates due to fading, noise, interference from other sources and mobile host movement. Due to large error rate, packet losses due to corruption in the wireless link are more as compared to congestion losses. The corruption loss rate is the packet loss rate due to corruption. Packet sending rate is not adjusted when the corruption loss rate is low, but it is required to decrease the sending rate rapidly to improve the reliability when the corruption loss rate becomes higher. Otherwise, there will be more lost packets due to corruption and more packets will be retransmission, responding the poor transmission reliability and more energy consumption of mobile hosts.

IV. PERFORMANCE EVOLUTION OF TCP VARIANTS

In the very earliest achievement of TCP, little was done to minimize the network congestion. Implementation used cumulative positive...
Acknowledgements and the expiry of a retransmit timer to provide reliability based on a simple go-back-n model. Several succeeding versions of TCP based on congestion control and avoidance mechanism have been developed, nowadays. In this section, we argue the performance of various TCP versions like regarding Tahoe, Reno, New Reno, SACK and Vegas.

A. TCP Tahoe
One of the TCP congestion control algorithms, is TCP Tahoe described, adds some new and enhance the earlier TCP implementation, including slow start, congestion avoidance and fast retransmission. This enhancement comprises change in round-trip-time estimation used to position retransmission time out values. TCP Tahoe fast retransmission algorithm outperforms the most when the packets are lost due to congestion. Sender should be waiting for retransmission timer to expire in the without fast retransmit algorithm. Hence, fast retransmission can save numerous seconds every time packet loss occurs, and the throughput is improved, consequently. The shortcoming in TCP Tahoe is that packet loss is detected after the whole timeout interval. When a packet loss is detected, TCP Tahoe performance becomes slow. Due to this reason transmission flow decreases rapidly. While fast retransmit makes Tahoe perform significantly better than a TCP implementation in which means of loss detection are merely the retransmission timers. It obtains significantly less than optimal performance on high delay bandwidth connections because of its initiation of slow start (which TCP Reno conversed in following section). Also, in the case of multiple losses within a single window, it is possible that the sender will retransmit packets which have already been delivered.

The problem with Tahoe is that it takes a complete timeout interval to detect a packet loss and in fact, in most implementations it takes even longer because of the coarse grain timeout. Also since it doesn’t send immediate ACK’s, it sends cumulative acknowledgements, there fore it follows a ‘go back n’ approach. Thus every time a packet is lost it waits for a timeout and the pipeline is emptied. This offers a major cost in high band-width delay product links.

B. TCP Reno
The Reno TCP mechanism is similar to the TCP Tahoe except it maintains improvements over Tahoe by adding to the fast recovery phase known as fast recovery algorithm. The significant improvement in TCP Reno in contrast to TCP Tahoe, prevent the communication path “pipe” from going empty after fast retransmit, and in that way it avoids slow start to fill it again after a packet loss. TCP Reno maintains the clocking of new data with duplicate ACKs which make it more beneficial than TCP Tahoe. In this way, TCP allows to directly cut its throughputs in half without the need for a slow start period to re-establish clocking between the data and ACKs. This improvement has the most noticeable effect on long delay-bandwidth connections where the slow start period lasts longer and large windows are needed to achieve optimal throughput. When a single packet is lost from a window of data, TCP Reno maintains it by fast recovery mechanism, in contrast when multiple packets are lost. Reno’s performance are same here as Tahoe. This indicates that if multiple packets are lost from the same window, TCP Reno almost immediately drag out of fast recovery, and stop until no new packet can be sent. The above discussion leads to conclusion that fast recovery mechanism introduced by TCP Reno handles multiple packet losses within a single window poorly.

In TCP Reno, the window size is cyclically changed in a typical situation. The window size continues to be increased until packet loss occurs. TCP Reno has two phases in increasing its window size: slow start phase and congestion avoidance phase. When an ACK packet is received by TCP at the sender side at time $t+\tau_A$ [sec], the current window size $cwnd (t+\tau_A)$ is updated from $cwnd (t)$ as follows:

$$cwnd (t+\tau_A) =$$

- Slow start phase:
  $$cwnd (t) + 1,$$ if $cwnd(t) < ssth(t)$;
  $$cwnd (t) + 1/cwnd(t)$$ if $cwnd(t) \geq ssth(t)$;

- Congestion avoidance phase:
  $$cwnd (t+\tau_A) = cwnd (t)$$

Where $ssth (t)$ [packets] is a threshold value at which TCP changes its phase from slow start phase to congestion avoidance phase. When packet loss is detected by retransmission timeout expiration, $cwnd (t)$ and $ssth (t)$ are updated as:

$$cwnd (t) = 1;$$
$$ssth (t) = cwnd(t)/2$$

On the other hand, when TCP detects packet loss by a fast retransmit algorithm, it changes $cwnd (t)$ and $ssth (t)$ as:

$$ssth (t) = cwnd(t)/2;$$
$$cwnd (t) = ssth(t)$$

C. TCP New Reno
The experimental version of TCP Reno is known as TCP New Reno. It is slightly different than TCP Reno in fast recovery algorithm. New Reno is more competent than Reno when multiple packets losses occur. New Reno and Reno both correspond to go through fast retransmit when multiple duplicate packets received, but it does not come out from fast recovery phase until all outstanding data was not acknowledged. It implies that in New Reno, partial ACK do not take TCP out of fast recovery but they are treated as an indicator that the packet in the sequence space has been lost, and should be retransmitted. Therefore, when multiple packets are
lost from a single window of data, at this time New Reno can improve without retransmission time out. The retransmitting rate is one packet loss per round trip time until all of the lost packets from that window have been transmitted. It exist in fast recovery till all the data is injected into network, and still waiting for an acknowledgement that fast recovery was initiated. The critical issue in TCP New Reno is that it is capable of handling multiple packet losses in a single window. It is limited to detecting and resending only one packet loss per round trip-time. This insufficiency becomes more distinct as the delay-bandwidth becomes greater. However, still there are situations when stalls can occur if packets are lost in successive windows, like all of the previous versions of TCP New Reno which infer that all lost packets are due to congestion and it may therefore unnecessarily cut the congestion window size when errors occur. There are some steps of congestion control for New Reno transmission control protocol.

Step 1: Initially
\[ cwnd = \min (4*\text{mss}, \max (2*\text{mss}, 4380 \text{ bytes})) \]
\[ \text{ss\_threshold} = \max (cwnd/2, 2*\text{MSS}) \]

Step 2: Slow Start Algorithm (Exponential Increases)
If (receive Acks & cwnd < ss\_threshold)
\[ cwnd = cwnd + 1; \]

Step 3: Congestion Avoidance Algorithm (Additive Increase)
If (receive ACKs) {
  If (cwnd > ss\_threshold)
    \[ cwnd = cwnd + \text{segsize} * \text{segsize} / cwnd; \]
  Else
    \[ cwnd = cwnd + 1; \]
}

Step 4: Congestion Detection Algorithm (Multiplicative Decrease): Fast Retransmission and Fast Recovery
If (congestion) {
  If (Receive same Acks 3 time or retransmission time out)
  \[ cwnd = cwnd/2; \]
  If (Retransmission time out)
    \[ cwnd = \text{initial}; \]
  Exit and call Slow Start step;
  Else /* Receive same Acks 3 time*/
    \[ cwnd = \text{SS\_threshold}; \]
  Exit and call congestion avoidance step;
}

D. TCP Vegas
TCP Vegas is an implementation of TCP based on TCP Reno with modifications in the sending side that try to achieve a more efficient use of available bandwidth. It was developed at the University of Arizona, and the authors claim that it achieves 40% to 70% better throughput with one-fifth to one-half the losses compared to TCP Reno. Vegas feature a new retransmission mechanism, a congestion avoidance mechanism, and a modified Slow Start.

The feature of Vegas that wireless ad hoc networks in particular could benefit from is the more efficient retransmission mechanism. It reads the system clock for each segment and thus calculates a more accurate RTT time. This mechanism allows it to measure a timeout with higher precision and retransmit without having to wait for three duplicate ACKS or the less precise, and thus longer, retransmission timer to time out. The authors of TCP Vegas found the average timeout interval of Reno to be more than three times higher the correct value that would be calculated with an accurate clock. This inaccuracy in Reno's timers introduces significant delays to retransmissions, lowering overall throughput. Vegas also adds another improvement; when a non-duplicate ACK is received and it is the first or second after a retransmission, Vegas checks if the time interval since the segment was sent is larger than the timeout value. If it is, it retransmits the segment without having to wait for a duplicate ACK to arrive.

TCP Vegas controls its window size by observing RTTs (Round Trip Time) of packets that the sender host has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles the window size. If RTTs become small, on the other hand, the sender host of TCP Vegas determines that the network is relieved from the congestion, and increases the window size again. Hence, the window size in an ideal situation is expected to be converged to an appropriate value.

That is,
\[ cwnd(t + t A) = \begin{cases} cwnd (t) + 1, & \text{ifdiff} < \text{a/base\_rtt} \\ cwnd (t), & \text{if} a/\text{base\_rtt} \leq \text{diff} = b/\text{base\_rtt} \\ cwnd (t) - 1, & \text{if} b/\text{base\_rtt} < \text{diff} \end{cases} \]
\[ \text{diff} = cwnd(t)/\text{base\_rtt} - cwnd(t)/\text{rtt} \]

Where \( \text{rtt} \) [sec] is an observed round trip time, \( \text{base\_rtt} \) [sec] is the smallest value of observed RTTs, and \( \text{a} \) and \( \text{b} \) are some constant values.

E. SACK TCP
The objective of SACK TCP [9, 10] is to alleviate TCP’s poor performance when multiple packets are lost from one window of data. SACK TCP defines TCP’s selective acknowledgement option, which allows the receiver to inform the sender which packets have been received, thereby giving the sender a better picture of the receiver's buffer. The sender can then retransmit only the missing data segments. Without SACK, the sender is forced to either wait a roundtrip time to find out about each lost packet, or to...
retransmit segments which have already been correctly received, generating needless retransmissions. It is possible for the receiver to send up to four SACK blocks in each segment, although if the timestamp option is used for Round Trip Time Measurement (RTTM), this is reduced to three SACK blocks. The receiver must follow these rules if it chooses to send a SACK option:

1. The first SACK block must specify the block of data containing the segment which triggered this ACK, unless that segment advanced the Acknowledgement Number field in the header.
2. Include as many distinct SACK blocks as possible in the SACK option.
3. The SACK option should be filled out by repeating the most recently reported SACK blocks. After the first SACK block, the following SACK blocks may be listed in arbitrary order.

The redundancy of blocks in the SACK option increases the robustness in the presence of lost ACKs; if one ACK is lost, the next ACK will contain overlapping information.

V. TCP VARIANTS COMPARISON STUDY

<table>
<thead>
<tr>
<th>Algorithms/ TCP Variants</th>
<th>TCP Tahoe</th>
<th>TCP Reno</th>
<th>TCP New Reno</th>
<th>TCP SACK</th>
<th>TCP Vegas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow Start</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>UV</td>
</tr>
<tr>
<td>Congestion Avoidance</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>UV</td>
</tr>
<tr>
<td>Fast Retransmit</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Fast Recovery</td>
<td>No</td>
<td>Yes</td>
<td>UV</td>
<td>UV</td>
<td>Yes</td>
</tr>
<tr>
<td>Retransmission Mechanism</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N M</td>
</tr>
<tr>
<td>Congestion Control Mechanism</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N M</td>
</tr>
<tr>
<td>Selective ACK Mechanism</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1 shows a summary of comparative study of different variants of TCP.

From the survey of different literatures, it has been observed that TCP Reno has a fast recovery mechanism. TCP NewReno and TCP Sack have an updated mechanism for recovery. But TCP Vegas has a new mechanism for Retransmission and Congestion Control with updated mechanism of recovery, as shown in table 1.

Due to these new mechanisms TCP Vegas can transmit more packets than TCP Reno and TCP New Reno as shown in figure 2. TCP Vegas also perform well when random loss rate increases. From figure 3 it can be observe, Goodput calculated for TCP Vegas is comparatively better than other.

CONCLUSION

This paper presents a study of five recently TCP protocols optimized for wireless networks, TCP Tahoe, TCP Reno, TCP New Reno, TCP SACK, TCP Vegas. This paper concluded that congestion is the main problem in different variants of TCP. Each variant has different mechanism to control congestion in network. But TCP Vegas has improved mechanism to recover the loss of packet due to congestion and corruption in network. It has improved mechanism for Slow Start and Congestion Avoidance and also included new mechanism for Retransmission and Congestion Control. So it can be analyzed that TCP Vegas performs better than other TCP variants.

REFERENCES

[1] Imrich Chlamtac, Marco Conti, Jennifer J.-N. Liu .Mobile ad hoc networking: imperatives and challenges, School of Engineering, University of Texas at Dallas, Dallas, TX, USA ,b Istituto IIT, Consiglio Nazionale delle Ricerche, Pisa, Italy ,c Department of Computer Science, University of Texas at Dallas, Dallas, TX, USA. Ad Hoc Networks 1 (2003)

Study of TCP Variants over Wireless Network


