International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-2 Number-1 Issue-3 March-2012

Performance Improvement of TCP by TCP Reno and SACK Acknowledgement

Reena Rai[1], Maneesh Shreevastava[2]
Department of Information Technology LNCT Bhopal, (M.P.) India[1,2]

Abstract

Transmission Control Protocol (TCP) is the dominating end-to-end transport layer protocol which provides secure and reliable data transfer together with some other protocols. In this review paper, we contend that existing approaches to improve TCP performance over mobile ad-hoc networks have focused only on a subset of the factors affecting TCP performance by TCP Reno, SACK and Vegas. Effective resource utilization, such as bandwidth utilization, retransmission rate and window size, is compared. For evaluate these TCP congestion control algorithms from many aspects are present and we also concern fair resource allocation from two main categories, one is fairness between different delay links, and the other is competition between different TCP congestion control algorithms.

Keywords

TCP Reno, SACK and Vegas, TCP, MANETs, Wireless, routing protocol, data transmissions, destination, TCP performance, TCP's timers.

1. Introduction

Early TCP implementation uses go-back-n model with cumulative positive acknowledgement and requires a retransmit time-out to retransmit the lost packet. These TCP did little to minimize network congestion. The operation of TCP in wireless/mobile communications has been an important research issue in recent years, owing to the impressive growth experienced in that area of modern telecommunications during the past decade. Significant contributions, such as the one presented in [1], indicate that the unmodified, standardized operation of TCP is not well aligned with the peculiarities of cellular environments. Terminal movement across cell boundaries, leading to handover, is misinterpreted by common TCP implementations as sign of congestion within the fixed network. To handle such congestion, TCP unnecessarily slows down transmission by reducing window sizes, and performing retransmissions, if relevant need arises.

In our Paper, we will evaluate the congestion control algorithms in Reno, Vegas and SACK TCP from different aspects. First, we will compare the performance of these algorithms: how much of the available network bandwidth does it utilize? How frequently does it retransmit packets? How does it modify window size on congestion? These comparisons are based on each version TCP running separately on a congested network. The second evaluation is the fairness of sharing the network. This comparison is taken in two categories of experiment. One is the fairness between different delay connections running the same version TCP. Some algorithms may bias against long delay connection, such as Reno TCP and SACK. The other experiment is carried out between different versions TCP when they compete each other on the same connection. TCP Vegas does not receive a fair share of bandwidth when competing with other TCP Reno or SACK connections. Since bias exists in both categories, how different queue algorithms may affect the fairness is also studied.

We shall assume that packet losses due to network loss are minimal and most of the packet losses are due to buffer overflows at the router. Thus it becomes increasingly important for TCP to react to a packet loss and take action to reduce congestion. TCP ensures reliability by starting a timer whenever it sends a segment. If it does not receive an acknowledgement from the receiver within the 'time-out' interval then it retransmits the segment. We shall start the paper by taking a brief look at each of the congestion avoidance algorithms and noting how they differ from each other. In the end we shall do a head to head comparison to further bring into light the differences.

There are several forms of acknowledgement:
Positive Acknowledgement:
The receiver explicitly notifies the sender which packets, messages, or segments were received correctly which may implicitly inform the sender

which packets were not received even though they were sent and thus may need to be retransmitted. Positive Acknowledgment with Re-Transmission (PAR), is a method used by TCP to verify receipt of transmitted data. PAR operates by re-transmitting data at an established period of time until the receiving host acknowledges reception of the data.

Negative Acknowledgment (NACK)
The receiver explicitly notifies the sender which packets, messages, or segments were received incorrectly and thus may need to be retransmitted.

Selective Acknowledgment (SACK)
The receiver explicitly lists which packets, messages, or segments in a stream are acknowledged (either negatively or positively). Positive selective acknowledgment is an option in TCP that is useful in Satellite Internet access.

Cumulative Acknowledgment
The receiver acknowledges that it correctly received a packet, message, or segment in a stream which implicitly informs the sender that the previous packets were received correctly. TCP uses cumulative acknowledgment with its TCP sliding window.

Retransmission is a very simple concept
Whenever one party sends something to the other party, it retains a copy of the data it sent until the recipient has acknowledged that it received it. In a variety of circumstances, e.g.:

if no such acknowledgment is forthcoming within a reasonable time, the time-out. The sender discovers, often through some out of band means, that the transmission was unsuccessful.
if the receiver knows that expected data has not arrived, and so notifies the sender.
if the receiver knows that the data has arrived, but in a damaged condition, and indicates that to the sender, the sender simply automatically retransmits the data.

2. Related Work

In this Research Paper [2] they implemented Multipath routing algorithms for heterogeneous network. Multipath routing separates the traffic among different paths to minimize congestion in terms of multiple alternative paths through a network which can provide a variety of benefits such as minimize delay and congestion, maximize bandwidth, or improved security. We propose a newly improved QoS multipath routing algorithm for heterogeneous networks. Different types of adhoc routing protocols are discussed in this paper such as Ad-Hoc On-Demand Distance Vector (AODV) , Ad-Hoc On Demand Multipath Distance Vector (AOMDV),QoS Ad-Hoc On Demand Multipath Distance Vector (QAOMDV), AOMDV is the extension of AODV routing protocol. QAOMDV is QoS version of AOMDV. These routing protocols are used in wireless network which is designed to form multiple routes from source to the destination and also avoid the loop formation so that it reduces congestion in the channel. The performance of AODV, AOMDV, and QAOMDV protocols are compared and proved the new routing protocol is better than others. The NS2 simulation result shows that improved performance of the heterogeneous network for newly proposed multipath routing protocol. The QAOMDV works better than other protocols in terms of delay, bandwidth, load balance, outing overhead and packet delivery ratio have been considered by varying the traffic load in the network. This paper analyzes the performance of different multi-path routing algorithms such as AODV, AOMDV and QAOMDV routing algorithms for wireless segment of heterogeneous network has been compared. The heterogeneous network is the combination of fixed and mobile network. Multipath routing protocols that computes multiple paths during route discovery avoids high overhead, latency and bandwidth. It is observed the performance of QAOMDV, a QoS multipath routing protocol of AOMDV, is efficient than DSR, AODV, AOMDV and DSDV.

Their Simulation results shows that the performance of QAOMDV is better than other routing protocol in wireless side and hierarchical routing is used in wired network. They proved that Multipath routing algorithm provides low delay and high throughput, better bandwidth utilization and low packet loss during data transmission. Finally the Timing analysis gives the comparison between different traffic pattern and Different routing protocols are compared by Average End to End delay with pause time.

3. Proposed Technique

TCP congestion control lies in Additive Increase Multiplicative Decrease (AIMD), halving the congestion window for every window containing a packet loss, and increasing the congestion window by roughly one segment per RTT otherwise. and TCP congestion control is the Retransmit Timer, including the exponential bakeoffs of the retransmit timer when a retransmitted packet is itself dropped. The third fundamental component is the Slow-Start mechanism for the initial probing for available bandwidth. The fourth TCP congestion control mechanism is ACK-clocking, where the arrival of acknowledgements at the sender is used to clock out the transmission of new data.

The TCP variants discussed in this paper, except TCP Vegas, all adhere to this underlying framework of Slow-Start, AIMD, Retransmit Timers, and ACK-clocking. None of these changes alter the fundamental underlying dynamics of TCP congestion control. Instead, these changes help to avoid unnecessary Retransmit Timeouts, correct unnecessary Fast Retransmits and Retransmit Timeouts resulting from disordered or delayed packets, and reduce unnecessary costs (in delay and unnecessary retransmits) associated with the mechanism of congestion notification.

TCP congestion control
Main algorithms
Slow start
Congestion Avoidance
Fast Retransmit
Fast Recovery
TCP SACK (Selective Acknowledgement)

TCP Tahoe
The Tahoe TCP implementation added a number of new algorithms and refinements to earlier TCP implementations. The new algorithms include Slow-Start, Congestion Avoidance, and Fast Retransmit [3]. The refinements include a modification to the round-trip time estimator used to set retransmission timeout values. The Fast Retransmit algorithm is of special interest because it is modified in subsequent versions of TCP. With Fast Retransmit, after receiving a small number of duplicate acknowledgments for the same TCP segment (dup ACKs), the data sender infers that a packet has been lost and retransmits the packet without waiting for a retransmission timer to expire, leading to higher channel utilization and connection throughput [4].

TCP Reno
The new algorithm prevents the communication channel from going empty after Fast Retransmit, thereby avoiding the need to Slow-Start to re-fill it after a single packet loss. The Reno TCP implementation retained the enhancements incorporated into Tahoe TCP but modified the Fast Retransmit operation to include Fast Recovery [5]. Fast Recovery operates by assuming each dup ACK received represents a single packet having left the pipe. Thus, during Fast Recovery the TCP sender is able to make intelligent estimates of the amount of outstanding data. A TCP sender enters fast Recovery after receiving an initial threshold of dup ACKs. Once the threshold of dup ACKs is received, the sender retransmits one packet and reduces its congestion window by one half. After entering Fast Recovery and retransmit a single packet, the sender effectively waits until half of a window of 2 dup ACKs have been received, and then sends a new packet for each additional dup ACK that is received. Upon receipt of an ACK for new data, the sender exits Fast Recovery. Reno significantly improves upon the behavior of Tahoe TCP when a single packet is dropped from a window of data, but can suffer from performance problems when multiple packets are dropped from a window of data.

TCP SACK
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 congestion control algorithms implemented in SACK TCP are a 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 (Selective Acknowledgement) to TCP does not change the basic underlying congestion control algorithms. 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. During Fast Recovery, SACK maintains a variable called pipe that represents the estimated number of packets outstanding in the path. The sender only retransmits data when estimated number of packets in the path is less than the congestion window. Use of the pipe variable decouples the decision of when to send a packet from the decision of which packet to send. The sender maintains a data structure that remembers acknowledgments from previous SACK options. When the sender is allowed to send a packet, it retransmits the next packet from the list of packets inferred to be missing at the receiver. The SACK sender has a special handling for partial ACKs (ACKs

received during Fast Recovery that advance the Acknowledgment Number field of TCP header, but do not take the sender out of fast Recovery). The sender decrements pipe by two rather than one for partial ACKs, the SACK sender never recovers more slowly than a Slow-Start. Detailed description of SACK TCP can be found in [6].

TCP Vegas
The idea is that when the network is not congested, the actual flow rate will be close to the expected flow rate. Otherwise, the actual flow rate will be smaller than the expected flow rate. TCP Vegas adopts a more sophisticated bandwidth estimation scheme. It uses the difference between expected and actual flow rates to estimate the available bandwidth in the network. TCP Vegas, using this difference in flow rates, estimates the congestion level in the network and updates the window size accordingly. This difference in the flow rates can be easily translated into the difference between the window size and the number of acknowledged packets during the round trip time, using the equation TCP Vegas tries to keep at least α packet but no more than β packets in the queues.

The reason behind this is that TCP Vegas attempts to detect and utilize the extra bandwidth whenever it becomes available without congesting the network. This mechanism is fundamentally different from that used by TCP Reno. TCP Reno always updates its window size to guarantee full utilization of available bandwidth, leading to constant packet losses, whereas TCP Vegas does not cause any oscillation in window size once it converges to an equilibrium point [7]. Our paper is focused on Reno, SACK and Vegas TCP since Tahoe is replaced by Reno in most of today's applications.

Congestion Window Size Variation
One main difference in congestion control algorithms of TCP SACK and TCP Reno is how they deal with more than one packet loss in one congestion window. We try simulating the case when four packets are dropped in one congestion window to see the window size variation. i hope when congestion window of TCP Reno drops to 0 and slow-start and more than one packet are dropped in one window. Congestion window of TCP Vegas oscillates when more packets are dropped, but never goes back from slow-start. TCP SACK maintains the same window size as the value after the first packet drop and returns to a higher window size than both Reno and Vegas. The algorithm of TCP SACK performs better in the case of more than one packet is dropped in one window.

4. Performance Evaluation

Justify the Behaviour on Long Delay connections:
To justify the observation in [7] that TCP Reno is biased against the connections with longer delays. The reason for this behaviour is as follows. While a source does not detect any congestion, it continues to increase its window size by one during one round trip time (RTT). Obviously, connections with a shorter delay can update their window sizes faster than those with longer delays, and thus capture higher bandwidths. To our understanding, TCP SACK does not change this window increasing mechanism, so we expect the same unfair behaviour with TCP SACK. We try to designing the simulation scenarios as follows. The network topology is shown in Topology fig 1. S1 and S2 will be set to be the same TCP agents, such as two Reno, two Vegas or two SACK TCP agents, respectively. Results of X=1ms (the same propagation delay as comparison baseline) and X=23ms (the RTT of longer delay connection is 8 times of the shorter one) will be collected to show the fairness between different delay connections.

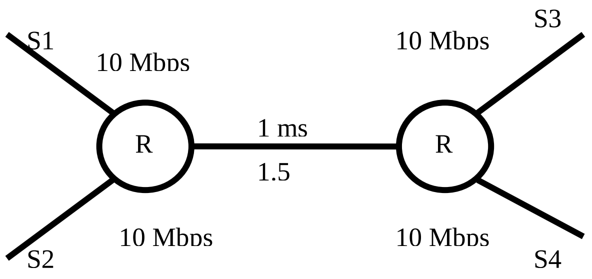

Fig 1. Topology Network

5. Conclusion

In this research paper, we propose to improve the performance of TCP Reno, TCP Vegas and TCP SACK from many aspects. of the both TCP Vegas and TCP SACK make some performance improvements to TCP Reno. TCP Vegas achieves higher throughput than Reno and SACK for large loss rate. TCP SACK is better when more than one packet is dropped in one window. TCP Vegas causes much fewer packets retransmissions than TCP Reno and SACK.

We also suggest a change in Vegas algorithm to make Vegas more aggressive in the competition. This may be worthy of further investigation. The efforts in analysis of queuing algorithms effects lie in the gateway side of the network. There are many suggestions of modification that lie on the host side to improve the fairness.

References

[1] R. Caceres, and Iftode. "Improving the performace of reliable transport protocols in mobile computing Environments", IEEE JSAC, Vol 13, No 5, June 1995.

[2] S.Santhi, G.Sudha Sadasivam, "Performance Evaluation of Different Routing Protocols to Minimize Congestion in Heterogeneous Network", IEEE-International Conference on Recent Trends in Information Technology, pp. 336-341, 2011.

[3] V. Jacobson, Congestion avoidance and control, ACM SIGCOMM Computer Communication Review, v.18 n.4, p.314-329, August 1988.

[4] Kevin Fall, Sally Floyd, Simulation-based comparisons of Tahoe, Reno and SACK TCP, ACM SIGCOMM Computer Communication Review, v.26 n.3, p.5-21, July 1996.

[5] V. Jacobson. "Modified TCP Congestion Avoidance Algorithm", Technical report, 30 Apr. 1990.

[6] Kevin Fall, Sally Floyd, Simulation-based comparisons of Tahoe, Reno and SACK TCP, ACM SIGCOMM Computer Communication Review, v.26 n.3, p.5-21.

[7] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand, Analysis and Comparison of TCP Reno and Vegas.

Reena rai was born in korba dist. Korba,(Chhattisgarh) India on 26th September 1984. She received his Bachelor Of Engineering Degree in Information Technology with first division and M-Tech in Information Technology. Her research interests include Computing Techniques, Security System , Robotics and Environmental with knowledge and skills for growth and development of Nation.