Progress Of Different TCP Variant

*Krishan Bhatia †Ajinkya Valanjoo,
* Assistant professor (Electronics and Tele-Comm)S.S.J.C.E.T. Mumbai, India
†P.G. Student (Electronics and Tele-Comm) V.E.S. Institute of Technology Mumbai,India

***Abstract*—At the beginning of development of network technology TCP were designed assuming that communication is using wired network, but recently there is huge demand and use of wireless network for communication. TCP protocol using wireless network unable to handle causes of packet losses in wireless network and also causes of unnecessary delay .So the biggest challenge in MANET is design of TCP variant which should give best performance in all scenarios. Uptil now more than dozens of variants designed and modified by research community but not a single TCP variant performance well in all scenarios. Such as congestion, link failure, signal loss, interference as well as rod and grid network topologies. As some of TCP-variant performs well in particular scenarios but degrades in other scenarios. Our objective was to design such a TCP variant which has to perform optimum in every network scenario For Example congestion, link failure, signal loss and interference scenario as well as chain and topologies.**

***Index Terms*—TCP, MANET.**

I. Introduction

A. TCP and Mobile Ad-Hoc Networks

Transmission Control Protocol (TCP) [1] is the predominant Internet protocol and it carries approximately 90 % of Internet traffic in todays heterogeneous wireless and wired networks.TCP is reliable end to end protocol because TCP is trying to provide reliable data transmission between two entities. TCP is widely used as a connection oriented transport layer protocol that provides reliable data packet delivery over unreliable links.TCP primary purpose is to provide a connection oriented reliable data transfer service between different applications to be able to provide these services on top of an unreliable communication system. TCP needs to consider data transfer, reliability flow control, multiplexing, TCP segment, and congestion control and connection management. TCP does not depend on the underlying network layers and, hence, design of various TCP variants is based on the properties of wired networks.

TCP consists of a set of rules: for the protocol, that are used with the Internet Protocol, and for the IP, to send data ”in a form of message units” between computers over the Internet. While IP handles actual delivery of the data, TCP keeps track of the individual units of data transmission, called segments that a message is divided into for efficient routing through the network.

MANET is considered as promising communication network in situations where rapid deployment and self-configuration are essential. In ad hoc networks, nodes are allowed to communicate with each other without any existing infrastructure[1].

Here every node should also play the role of a router. This kind of networking can be applied to scenarios like conference room, disaster management, and battle field communication and places where deployment of infrastructure is either difficult or costly.

Opposed to infrastructure wireless networks, where each user directly communicates with an access point or base station, a mobile ad hoc network, or MANET, does not rely on a fixed infrastructure for its operation. The network is an autonomous transitory association of mobile nodes that communicate with each other over wireless links. Nodes that lie within each others send range can communicate directly and are responsible for dynamically discovering each other. In order to enable communication between nodes that are not directly within each others send range, intermediate nodes act as routers that relay packets generated by other nodes to their destination. These nodes are often energy constrained that is, battery-powered devices with a great diversity in their capabilities. Furthermore, devices are free to join or leave the network and they may move randomly, possibly resulting in rapid and unpredictable topology changes. In this energy-constrained, dynamic, distributed multi-hop environment, nodes need to organize themselves dynamically in order to provide the necessary network functionality in the absence of fixed infrastructure or central administration.

B. Problems in Wireless Network

In ad hoc networks, the principal problem of TCP lies in performing congestion control in case of losses that are not induced by network congestion. Since bit error rates are very low in wired networks, nearly all TCP versions nowadays assume that packets losses are due to congestion. Consequently, when a packet is detected to be lost, either by timeout or by multiple duplicated ACKs, TCP slows down the sending rate by adjusting its congestion window. Unfortunately, wireless networks suffer from several types of losses that are not related to congestion, making TCP not adapted to this environment[2].

Literature survey of TCP over Ad Hoc Networks we identified following problems.

In Wireless ad-hoc networks nodes may be mobile therefore no predefined topology as nodes can joins and leaves network so accordingly topology may change. When the topology of the network changes every time, then routing mechanism needs to trigger to find alternative roots to do the reliable end to end communication between sender and receiver.

Thus due to frequently changes in topology communication links may failures and there will be the loss of data segment. To recover segment loss TCP sender reduces sending rate by triggering congestion control mechanism which sets size of congestion window of its lowest value. Assuming that the loss of packet is due to congestion in network which is totally misjudged and there will be the underutilization of available bandwidth. Hence the performance of TCP degrades. These problems are due to lossy channels, Hidden and exposed

stations, path asymmetry which may appear in several forms like BW asymmetry, loss rate asymmetry and route asymmetry.

The paper is organised as follows. Section I provides Introduction. Section II Describes the Standard TCP congestion control algorithms Section III describes the various TCP variants IV Conclusion And Section V Future Work

II. TCP CONGESTION CONTROL ALGORITHM

A. Slow start

Slow start is conducted in the beginning of every TCP connection and its main purpose is to find the maximum available bandwidth at which it can send data without ca using the network to be congested. To realize this, slow start forces the TCP sender to transmit at a slow sending rate and then rapidly increasing it until the available bandwidth between the hosts is believed to be found[3].

Slow Start Algorithm
Initial: cwnd = 1;
For (each packet Acked)
cwnd++;
Until (congestion event/ cwnd≥ssthresh)

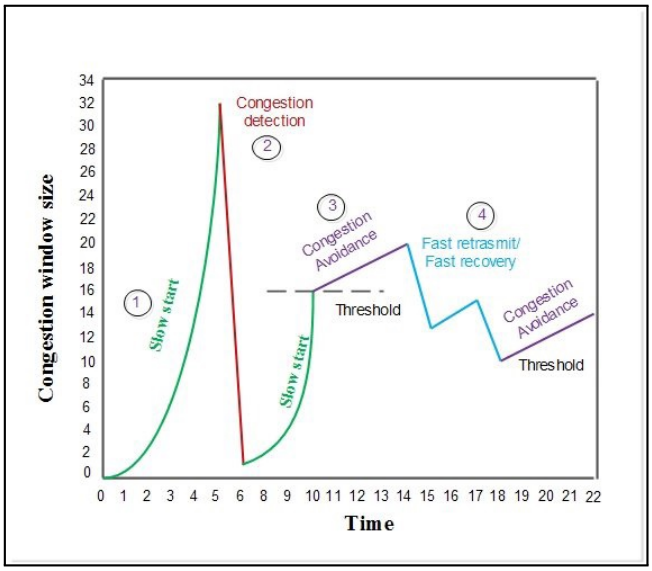

Fig. 1. Congestion Window Trace Format

B. Congestion avoidance

If the receiver window is large enough, the slow start mechanism described in the previous routers in between the hosts will start discarding packets. As mentioned earlier TCP interprets packet loss as a sign of congestion, and when this happens TCP invokes the Congestion Avoidance mechanism. Even though slow start and congestion avoidance is two different mechanisms they are more easily described together. In the joint description below a new TCP variable is introduced. This variable, ssthresh, is the slow start threshold which TCP uses to determine if slow start or congestion avoidance is to be conducted[3].

Congestion Avoidance Algorithm:
/* slow start is over and cwnd≥ssthresh */
Every Ack:
cwnd = cwnd + (1/cwnd)
Until (Timeout or 3 DUPACKs)

C. Fast Retransmit

If an out-of-order segment is received TCP generates a so called duplicate acknowledgment. This duplicate acknowledgment is sent immediately from the receiver to the sender indicating that a segment arrived out-of-order, and which segment that was supposed to be received. Since it is not possible to know whether the duplicate acknowledgment was caused by a lost segment or just reordering of segments, the sender waits for three duplicate acknowledgments before retransmitting the segment. If this limit would have been lower, this would increase the chance of reordered segments causing duplicates to be created, and transmitted needlessly. The advantage of this mechanism is that TCP does not have to wait for the retransmission timer to expire. It simply assumes that three duplicate acknowledgments is a good indicator of a lost segment[3].

Fast Retransmit Algorithm:
/* After receiving 3 DUPACKs */
Resend lost packet;
Invoke Fast Recovery algorithm

D. Fast Recovery

After fast retransmit is conduct congestion avoidance and not slow start is performed. This behavior iscalled Fast Recovery[1]. Fast recovery is an algorithm allows for higher throughput under congestion,especially when using large congestion windows. Receiving three duplicate acknowledgments tells TCP more than the expiration of the retransmission timer. Since the receiving, TCP only can generate duplicate acknowledgments when it is receiving other segments it is an indication that data still flows between the different hosts, and that the congestion is not that severe. By using this approach, skipping the slow start, the TCP does not reduce the transfer rate unnecessarily much[3].

Fast Recovery Algorithm:
/* after fast retransmit; do not enter slow start */
ssthresh = cwnd / 2;
cwnd = ssthresh + 3;
Each DACK received;
cwnd ++;
Send new packet if allow;
After receiving an Ack:
If partial Ack;
Stay in fast recovery;
Retransmit next lost packet (one packet per RTT);
If Full Ack;
cwnd = ssthresh;
Exit fast recovery;
Invoke Congestion Avoidance Algorithm;
When Timeout:
ssthresh = cwnd /2;
cwnd = 1;
Invoke Slow Start Algorithm;

III. TCP VARIANTS

A. TCP Tahoe

Early TCP implementations followed a go-back- model using cumulative positive acknowledgment and requiring a retransmit timer expiration to re-send data lost during transport[4]. In the process of trying to improve the original TCP three traffic management mechanisms slow start, congestion avoidance and fast retransmit were introduced to the original TCP and the new TCP version was named as TCP Tahoe.TCP Tahoe is the TCP variant developed by Jacobson in 1988. It uses Additive Increase Multiplicative Decrease (AIMD) algorithm to adjust window size. It means that increases the congestion window by one for successful packet delivery and reduces the window to half of its actual size in case of data loss or any delay only when it receives the first negative acknowledge. In case of timeout event, it reduces congestion window to 1 MSS[5].

TCP Tahoe uses packet loss probability to adjust the congestion window size.

During Slow Start stage, TCP Tahoe increases window size exponentially i.e. for every acknowledgement received, it sends two packets.

During Congestion Avoidance, it increases the window size by one packet per Round Trip Time (RTT) so as to avoid congestion.

In case of packet loss, it reduces the window size to one and enters in Slow Start stage.

Due to automatic set back to slow start mode of operation with initial congestion window of one every time packet loss is detected we see TCP Tahoe does not prevent the communication link from going empty. Hence this may have high cost in high bandwidth product links[4].

B. Tcp Reno

TCP Reno retains the basic principle of Tahoe, such as slow starts and the coarse grain retransmit timer. However it adds some intelligence over it so that lost packets are detected earlier and the pipeline is not emptied every time a packet is lost. Reno requires that we receive immediate acknowledgement whenever a segment is received. The logic behind this is that whenever we receive a duplicate acknowledgment, then his duplicate acknowledgment could have been received if the next segment in sequence expected, has been delayed in the network and the segments reached there out of order or else that the packet is lost. If we receive a number of duplicate acknowledgements then that means that sufficient time have passed and even if the segment had taken a longer path, it should have gotten to the receiver by now. There is a very high probability that it was lost. So Reno suggests FastRe-Transmit. Whenever we receive 3 duplicate ACKs we take it as a sign that the segment was lost, so we re-transmit the segment without waiting for timeout. Thus we manage to re-transmit the segment with the pipe almost full. Another modification that RENO makes is in that after a packet loss, it does not reduce the congestion window to 1. Since this empties the pipe. It enters into an algorithm which we call Fast-Recovery.

Reno performs very well over TCP when the packet losses are small. But when we have multiple packet losses in one window then RENO doesnt perform too well and its performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect a single packet loss. If there is multiple packet drops then the first info about the packet loss comes when we receive the duplicate ACKs. But the information about the second packet which was lost will come only after the ACK for the retransmitted first segment reaches the sender after one RTT[6].

Also it is possible that the CWD is reduced twice for packet losses which occurred in one window. Suppose we send packets 1, 2, 3, 4, 5, 6, 7, 8, 9 in that order. Suppose packets 1 and 2 are lost. The ACKs generated by 2, 4, 5 will cause the retransmission of 1 and the CWD is reduced to 7. Then when we receive ACK for 6, 7, 8, 9 our CWD is sufficiently large to allow to us to send 10, 11. When the re-transmitted segment 1 reaches the receiver we get a fresh ACK and we exit fast-recovery and set CWD to 4. Then we get two more ACKs for 2(due to 10, 11) so once again we enter fast-retransmit and re-transmit 2 and then enter fast recovery. Thus when we exit fast recovery for the second time our window size is set to 2. Thus we reduced our window size twice for packets lost in one window.

Another problem is that if the widow is very small when the loss occurs then we would never receive enough duplicate acknowledgements for a fast retransmit and we would have to wait for a coarse grained timeout. Thus is cannot effectively detect multiple packet losses.

C. TCP New Reno

The experimental version of TCP Reno is known as TCP New Reno[6]. 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[6] 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 delaybandwidth 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 dueto congestion and it may therefore unnecessarily cut the congestion window size when errors occur. There are[6] some steps of congestion control for New Reno transmission

control protocol.

D. Selective Acknowledgment (Sack)

The Selective Acknowledgment (SACK) mechanism, RFC2018, an extension to Transmission Control Protocols (TCP) ACK mechanism, allows a data receiver to explicitly acknowledge arrived out-of-order data to a data sender. When using SACKs, a TCP data sender need not retransmit SACK data during the loss recovery period. Previous research showed that SACKs improve TCP throughput when multiple losses occur within the same window. The success of SACK-based loss recovery algorithm is proportional to the SACK information received from the data receiver [7].

With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender need retransmit only the segments that have actually been lost [8].

TCP with Selective Acknowledgments is an extension of TCP Reno and it works around the problems face by TCP Reno and TCP New-Reno, namely detection of multiple lost packets, and re-transmission of more than one lost packet per RTT [9]. SACK retains the Slow-Start and Fast Re-Transmit parts of Reno. It also has the coarse grained timeout of Tahoe to fall back on, in case a packet loss is not detected by the modified algorithm. TCP-Sack requires that segments not be acknowledged cumulatively but should be acknowledged selectively. Thus each ACK has a block which describes which segments are being acknowledged. Thus the sender has a picture of which segments have been acknowledged and which are still outstanding. Whenever the sender enters fast recovery, it initializes a variable pipe which is an estimate of how much data is outstanding in the network, and it also set CWND to half the current size. Every time it receives an ACK it reduces the pipe by 1 and every time it retransmits a segment it increments it by 1. Whenever the pipe goes smaller than the CWD window it checks which segments are not received and send them. If there are no such segments outstanding then it sends a new packet. Thus more than one lost segment can be sent in one RTT.

Problem with SACK is that currently selective acknowledgements are not provided by the receiver to implement SACK well need to implement selective acknowledgment which is not a very easy task and Requires modification to the acknowledgement procedures at both sender and receiver sides.

E. Tcp Westwood

TCP Westwood proposes an end-to-end bandwidth estimation algorithm based on TCP Reno. TCP Westwood implements slow start and congestion avoidance phases as TCP Reno, but instead of halving the congestion window size as in TCP Reno when congestion happens, TCP Westwood adaptively estimates the available bandwidth and sets the congestion window size and slow start threshold accordingly to improve the link utilization. In TCP Westwood, packet loss is indicated by the reception of 3 duplicated acknowledgements (DUPACKs) or timeout expiration.

This sets the cwnd to 1 and ssthresh to BE after the timeout event and then the TCP Reno behavior continues. In TCP Westwood, the setting of ssthresh and cwnd is based on the bandwidth estimation, which is obtained by measuring the rate of the acknowledgments and collecting the information of the amount of packets delivered to the receiver in the ACK. Samples of bandwidth are computed as the amount of packet delivered divided by the inter-arrival time between two ACKs. Those sample bandwidth estimates are then filtered to achieve an accurate and fair estimation. TCP Westwood modifies the Additive Increase and Multiplicative Decrease (AIMD) in TCP Reno and adaptively sets the transmission rates to remove the oscillatory behavior of TCP Reno and to maximize the link utilizations. But this also causes TCP Westwood to degrade the performance of TCP Reno connections when they coexist in the network [10].

Problems of Tcp Westwood Perform poorly if it estimates incorrect bandwidth. Because of unpredictability in the behavior of the bandwidth estimation scheme used in TCP Westwood. Changes in the inter-arrival times of the acknowledgements cause improvement or worsening of the throughput in rather unpredictable ways. Additionally, the sensitivity of TCP Westwood Ackd Interval is variable.

F. TCP WestwoodNR

In TCP WestwoodNR [9] the sender continuously computes the connection Bandwidth Estimate (BWE) which is defined as the share of bottleneck bandwidth used by the connection. Thus, BWE is equal to the rate at which data is delivered to the TCP receiver. The estimate is based on the rate at which ACKs are received and on their payload. After a packet loss indication, (i.e, reception of 3 duplicate ACKs, or timeout expiration). The sender resets the congestion window and the slow start threshold based on BWE. More precisely, Cwin=BWE x RTT. To understand the rationale of TCP-WNR, note that BWE varies from flow to flow sharing the same bottleneck; it corresponds to the rate actually achieved by each INDIVIDUAL flow. Thus, it is a FEASIBLE (i.e. achievable) rate by definition. Consequently, the collection of all the BWE rates, as estimated by the connections sharing the same bottleneck, is a FEASIBLE set. When the bottleneck becomes saturated and packets are dropped, TCP-WNR selects a set of congestion windows that correspond exactly to the measured BWE rates and thus reproduce the current individual throughputs. The solution is feasible, but it is not guaranteed per se to be fair share. An additional property of this scheme, described in Section III, drives the rates to the same equilibrium point and makes it fair share under uniform propagation delays and single bottleneck assumptions. Another important element of this procedure is the RTT estimation. RTT is required to compute the window that supports the estimated rate BWE. Ideally, the RTT should be measured when the bottleneck is empty. In practice, it is set equal to the overall minimum round trip delay (RTTmin) measured so far on that connection (based on continuous monitoring of ACK RTTs).

G. Tcp Vegas

Bandwidth Estimation scheme used by TCP Vegas is more efficient than other TCP variants. This scheme makes bandwidth estimation by using the difference between the expected flow rates and the actual flow rates. TCP Vegas was introduced in 1994 as an alternative to TCP Reno and its implementation and tests showed that it achieves better throughput than TCP Reno. TCP Vegas bandwidth estimation differs from that in TCP Reno. Unlike TCP Reno, which uses packet loss as the indication of network congestion, TCP Vegas uses the difference between the estimated throughput and the measured throughput as the measure of congestion. TCP Vegas records the smallest measured round trip time as BaseRTT and computes the available bandwidth as: Expected-Bandwidth = WindowSize/ BaseRTT Where the WindowSize is the current window size. During the packet transmission, the round trip time (RTT) of packets are recorded. The actual throughput is calculated as: ActualBandwidth = WindowSize/ RTT The difference between the ExpectedBandwidth and ActualBandwidth is used to adjust the WindowSize: Diff = ExpectedBandwidth - Actual Bandwidth Two values α and β ($0 \leqslant \alpha \leqslant \leqslant$) are defined as the thresholds. If Diff $\leqslant \alpha$, the window size is increased during the next RTT; if Diff $\geqslant \beta$, then the window size is decreased during the next RTT. Otherwise, the window size is unchanged. The goal of TCP Vegas is to keep a certain number of packets or bytes in the queues of the network [11]. If the actual throughput is smaller than the expected throughput, TCP Vegas takes this as indication of network congestion, and if the actual throughput is very close to the expected throughput, it is suggested that the available bandwidth is not fully utilized, so TCP Vegas increases the window size.

This mechanism used in TCP Vegas to estimate the available bandwidth does not purposely cause any packet loss. Hence the oscillatory behavior is removed and a better throughput is achieved. Retransmission mechanism used by TCP-Vegas is more efficient as compared to TCP-Reno as it retransmits the corresponding packet as soon as it receives a single duplicate ACK and does not wait for three ACKs. TCP-Vegas as compared to TCP-Reno is more accurate and is less aggressive, thus it does not reduce its CWND unnecessarily.

It has problems when packets do not follow the same route and when large delays are present. When routes change for a certain TCP Vegas flow, the BaseRTT recorded from the previous route is no longer accurate; this affects the accuracy of ActualBandwidth and subsequently influences the performance of TCP Vegas. TCP Vegas could become unstable when there is large network delay for a flow; later established connections cannot get a fair share of the bandwidth, and when they coexist with TCP Reno connections, TCP Reno connections use most of the bandwidth.

H. FACK

The development in TCP SACK with Forward Acknowledgement is identified as TCP FACK [3]. The utilization of TCP FACK is almost identical to SACK but it establishes a little enhancement evaluated to it. It uses SACK option to better estimate the amount of data in transit. TCP FACK introduces a better way to halve the window when congestion is detected. When CWND is immediately halved, the sender stops transmitting for a while and then resumes when enough data has left the network. In this one RTT can be avoided when the window is gradually decreased .When congestion occurs; the window should be halved according to the multiplicative decrease of the correct CWND. Since the sender identifies congestion at least one RTT after it happened, if during that RTT it was in slow start mode, then the current CWND will be almost double than CWND when congestion occurred.

Therefore, in this case, CWND is first halved to estimate the correct CWND that should be further decreased.

IV. Conclusion

A detail review of existing TCP variants and its applicable algorithm are analyzed and describe about the protocol which one is better and suitable for packet and link utilization in the network congestion and link failure condition in Ad-hoc network environment because the traditional TCP treat all packet losses due to the congestion, it does not treat from the link failure.

The review are obtained and analyzed by the TCP variants: TCP Tahoe, TCP Reno, TCP New Reno, and TCP Vegas, TCP SACK, TCP Westwood,TCP WestwoodNR, TCP FACK . The most of protocol shows better uses and many of them shows poor responsiveness to changing network conditions and network utilization. Although there are various protocols and algorithms have been used, there is no single algorithm that can overcome the congested and unreliable nature of network. Here each and every variant has its own advantages and disadvantages to solve the networks problem s of TCP protocol.

In short, any protocol will be effective based on the parameters that are to be taken into consideration. To conclude this area is not completely explored to it maximum and still lot more research can be done towards establishing a basis for the development of new protocols.

V. Future Work

In previous Review Which had considered all the problems occurred in various TCP variants. Due to these problems throughput decreased, packet loss increased and delay occurred. Also considered difference between each variant. So, the aim of new TCP Variant is to increase the throughput with minimum packet loss and minimum delay as compared with other variants.

References

[1] JeroenHoebeke, Ingrid Moerman, Bart Dhoedt and Piet Demeester, An Overview of Mobile Ad Hoc Networks: Applications and ChallengesDepartment of Information Technology (INTEC),Ghent University

[2] Ahmad Al Hanbali, Eitan Altman, And Philippe Nain, Inria Sophia Antiplolis France, A SURVEY OF TCP OVER AD HOC NETWORKS, IEEE Communications Surveys and Tutorials ,Third Quarter 2005.

[3] SUN Xiaoling TCP Congestion Control Algorithm Research Computer and Information Engineering Institute Chafing University

[4] http://www.ukessays.com/ePssays/communications/transmission-control protocol

[5] Mrs.Alaa Ghaleb Seddik, TCP performance Study and Enhancement within wireless Multi-hop Ad-hoc network environments 30 March 2009.
[6] Dhananjay Bisen and Sanjeev Sharma IMPROVE PERFORMANCE OF TCP NEW RENO OVER MOBILE AD-HOC NETWORK USING ABRA International Journal of Wireless and Mobile Networks (IJWMN) Vol. 3, No. 2, April 2011
[7] Nasif Ekiz Abuthahir Habeeb Rahman Paul D. Amer Misbehaviors in TCP SACK Generation ACM SIGCOMM
[8] http://www.ietf.org/rfc/rfc2018.txt
[9] Claudio Casetti,Mario Gerla,SaverioMascolo,M.Y. Sanadidi And Ren Wang, TCPWestwood: End-to-End Congestion Control for Wired/Wireless NetworksWireless Networks 8, 467479, 2002
[10] Saleem-ullah Lar, Xiaofeng Liao and Songtao Guo, Modeling TCP NewReno Slow Start and Congestion- Avoidance using Simulation Approach, in IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, January 2011. Technical Report IAM-02-003, Univ. of Bern, July 2001.
[11] Seddik-Ghaleb, Y. Ghamri-Doudane, and S. M. Senouci, ”TCP WELCOME TCP Variant for Wireless Environment, Link losses, and COngestion packet loss ModEls, in First International Communication Systems and Networks and Workshops, COMSNETS 2009