Recovery of loss of packet in network using constant packet reordering

Sandeep Kumar Gonnade , Naresh Kumar Nagwani
CSE Department.
NIT Raipur (C.G.)
sandeep_gonnade@yahoo.co.in

Abstract-When the packet is reordered the most standard implementation of the Transmission control protocol (TCP) gives poor performance. This paper proposes a new version of the TCP which gives the high throughput when the packet reordering occurs and in another case if the packet reordering is not occurs then in that case also it is friendly to other version of the TCP. Transmission control protocol constant Packet Reordering (TCP-CPR) does not depend or rely on the duplicate acknowledgement to detect the packet loss. Instead the timer is used to maintain how long packet is transmitted. In this case timer is used to keep the track how long packets are transmitted. If acknowledgement are not received within the appropriate time then packet assume to loss because of the TCP-CPR does not depend on the duplicate acknowledgement. Packet reordering has does not effect on the performance of TCP-CPR.

Through the performance of the TCP-CPR consistently better than existing mechanism that make the try to make the TCP more robust to packet reordering. In case where packets are not reordered, it's verified that TCP-CPR maintains the same throughput as the typical implementation of TCP.

Keywords – TCP, TCP-CPR, congestion, packet, DUPACK

I. INTRODUCTION

As per the design format of the Transmission control protocol (TCP) error and congestion control mechanism which is based on the principle that packet loss is an indication of the network congestion. As per TCP senders backs off transmission rate by decreasing its congestion control windows. TCP uses two strategies for the detection of the packet loss the first one is based on the sender's retransmission timeout (RTO) which is also referred as coarse timeout. When the senders timeout which is responded by the congestion control by slow start which leads into decreasing congestion window to one segment. The packet detection loss is detected at the receiver side by using the sequence number. In this case receiver checks the sequence number of received packet. The hole in the sequence indicates that there is loss of the packet in such case the receiver generates the duplicate acknowledgement for every "out-of-order" segment it receives. Until the lost packet received, the entire reaming packet with higher sequence number is consider as out of order and will cause to creation of duplicates packets. After that sender retransmit the lost packet without waiting for timeout which helps to reduction of congestion windows. The main idea behind retransmit packet this is to improve the performance of TCP throughput by avoiding sender to timeout.

Using fast retransmit can continuously improve the TCP's performance in the presence of irregular reordering but it still operates under the assumption of that out – of – order packets which indicate the packet loss and which leads to congestion. As a result its performance degrades in the presence of persistent reordering. This is procedure for reordering both data and acknowledgment packet. Packet reordering is generally attributed to transient conditions pathological behavior and erroneous implementation.

I. EXISTING SYSTEM

The design of TCP's error and congestion control mechanisms was based on the premise that packet loss is an indication of network congestion. Therefore, upon detecting loss, the TCP sender backs off its transmission rate by decreasing its congestion window. TCP uses two strategies for detecting packet loss. The first one is based on the sender's retransmission timeout expiring and is sometimes referred to as coarse timeout. When the sender times out, congestion control responds by causing the sender to enter slow-start, drastically decreasing its congestion window to one segment. The other loss detection mechanism originates at the receiver and uses TCP's sequence number. Essentially, the receiver observes the sequence numbers of packets it receives; a "hole" in the sequence is considered indicative of a packet loss. Specifically, the receiver generates a "duplicate acknowledgment" (or DUPACK) for every "out-of-order" segment it receives. Note that until the lost packet is received, all other packets with higher sequence number are considered "out-of-order" and will cause DUPACKs to be generated. Modern TCP implementations adopt the fast retransmit algorithm which infers that a packet has been lost after the sender receives a few DUPACKs.

The sender then retransmits the lost packet without waiting for a timeout and reduces its congestion window in half. The basic idea behind fast retransmit is to improve TCP's throughput by avoiding the sender to timeout (which results in slow-start and consequently the shutting down of the congestion window to one).

Fast retransmit can substantially improve TCP's performance in the presence of sporadic reordering but it still operates under the assumption that out-of-order packets

indicate packet loss and therefore congestion. Consequently, its performance degrades.

Considerably in the presence of "constant reordering." This is the case for reordering of both data and acknowledgment packets. Indeed, it is well known that TCP performs poorly under significant packet reordering (which may not be necessarily caused by packet losses). Packet reordering is generally attributed to transient conditions, pathological behavior, and erroneous implementations. For example, oscillations or "route flaps" among routes with different round-trip times (RTTs) are a common cause for out-of-order packets observed in the Internet today. Internet experiments performed through MAE-East and reported in show that 90% of all connections tested experience packet reordering. Researchers at SLAC performed similar experiments and found that 25% of the connections monitored reordered packets. However, networks with radically different characteristics (when compared to the Internet, for example) can exhibit packet reordering as a result of their normal operation. Most standard implementations of TCP perform poorly when packets are reordered. In existing TCP was based on the premise that packet loss is an indication of network congestion. The exact lost packets in the network couldn't able to find. It causes redundancy of acknowledgement for packets.

Most standard implementations of TCP perform poorly when packets are reordered. In a existing TCP was based on the premise that packet loss is an indication of network congestion.

2.1 Limitations

- TCP detects packet loss through duplicate Acknowledgement.
- It performs poorly when packets are reordered.
- Its Throughput decreases whenever packet is reordered.
- Not easier to deploy.
- Decreased robustness.

I. PROPOSED SYSTEM

The basic idea behind TCP-CPR is to detect packet losses through the use of timers instead of duplicate acknowledgments. This is prompted by the observation that, under constant packet reordering, duplicate acknowledgments are a poor indication of packet losses. Because TCP-CPR relies solely on timers to detect packet loss, it is also robust to acknowledgment losses as the algorithm does not distinguish between data (on the forward path) or acknowledgment (on the reverse path) losses.

The proposed algorithms only require changes in the TCP sender and are therefore Backward-compatible with any TCP receiver. TCP-CPR's sender algorithm is still based on the concept of a congestion window, but the update of the congestion window follows slightly different rules than standard TCP. However, significant care was placed in making the algorithm fair with respect to other versions of TCP to ensure they can coexist Packets being processed by the sender are kept in one of two lists: the to-be-sent list contains all packets whose transmission is pending, waiting for an "opening" in the congestion window. The to-be-ack list contains those packets that were already sent but have not yet been acknowledged. Typically, when an application produces a packet it is first placed in the to-be-sent list; when the congestion window allows it, the packet is sent to the receiver and moved to the to-be-ack list; finally when an ACK for that packet arrives from the receiver, it is removed from the to-be-ack list (under cumulative ACKs, many packets will be simultaneously removed from to-be-ack). Alternatively, when it is detected that a packet was dropped, it is moved from the to-be-ack list back into the to-be-sent list.

As mentioned above, drops are always detected through timers. To this effect, whenever a packet is sent to the receiver and placed in the to-be-ack list, a timestamp is saved. When a packet remains in the to-be-ack list more than a certain amount of time it is assumed dropped. In particular, it assumed that a packet was dropped at time when exceeds the packet's timestamp in the to-be-ask list plus an estimated maximum possible round-trip time.

3.1 Advantages of Proposed system

- Proposed system works perfectly when packet is reordered.
- It uses Timer Control to detect the packet Loss.
- Its performance will be same even the packet is reordered.
- Proposed system not only depends on the duplicate acknowledgements and packet reordering to detect the packet losses.
- This system performs consistently better than existing mechanisms that try to make TCP more robust to packet reordering.
- Easier to deploy since no changes are required at the sender side.

II. PROBLEM FORMULATION MODULES

The Proposed system is divided into following main modules

- Transmission without Reordering
- Transmission with Packet Reordering
- segmentation
- Timer Control
- Packet Reordering
- Comparison chart

Module Description:

A. Transmission without Packet Reordering

If a message transmitted without packet reordering, then If part of a message is lost during the transmission then it's needed to retransmit the entire message or it's needed to retransmit from that particular part. Therefore, upon detecting loss, the TCP sender backs off its transmission rate by decreasing its congestion window. TCP uses two strategies for detecting packet loss. The first one is based on the sender's retransmission timeout (RTO) expiring and is sometimes referred to as coarse timeout. When the sender times out, congestion control responds by causing the sender to enter slow-start, drastically decreasing its congestion window to one segment. The other loss detection mechanism originates at the receiver and uses TCP's sequence number. Essentially, the receiver observes the sequence numbers of packets it receives; a "hole" in the sequence is considered indicative of a packet loss. Specifically, the receiver generates a "duplicate acknowledgment" (or DUPACK) for every "out-of-order" segment it receives. Note that until the lost packet is received, all other packets with higher sequence number are considered "out-of-order" and will cause DUPACKs to be generated. Modern TCP implementations adopt the fast retransmit algorithm which infers that a packet has been lost after the sender receives a few DUPACKs.

B. Transmission with Packet Reordering

If a message is transmitted as packets then it's needed to retransmit only the packet which is lost and not the entire message. The message is send from the source to the ingress router and then to the intermediate routers and then to the outgress router and the destination.

The basic idea behind TCP-CPR is to detect packet losses through the use of timers instead of duplicate acknowledgments. This is prompted by the observation that, under constant packet reordering, duplicate acknowledgments are a poor indication of packet losses. Because TCP-CPR relies solely on timers to detect packet loss, it is also robust to acknowledgment losses as the algorithm does not distinguish between data (on the forward path) or acknowledgment (on the reverse path) losses.

The proposed algorithms only require changes in the TCP sender and are therefore backward-compatible with any TCP receiver. TCP-CPR's sender algorithm is still based on the concept of a congestion window, but the update of the congestion window follows slightly different rules than standard TCP. However, significant care was placed in making the algorithm fair with respect to other versions of TCP to ensure they can coexist. Packets being processed by the sender are kept in one of two lists: the to-be-sent list contains all packets whose transmission is pending, waiting for an "opening" in the congestion window. The to-be-ack list contains those packets that were already sent but have not yet been acknowledged. Typically, when an application produces a packet it is first placed in the to-be-sent list; when the congestion window allows it, the packet is sent to the receiver and moved to the to-be-ack list; finally when an ACK for that packet arrives from the receiver, it is removed from the to-be-ack list (under cumulative ACKs, many packets will be simultaneously removed from to-be-ack). Alternatively, when it is detected that a packet was dropped, it is moved from the to-be-ack list back into the to-be-sent list.

C. Segmentation

Segmentation is the process of dividing the source code into small number of packets and transmitting the packets through the routers. It's defined certain limits for the size of the packets. The packet is sent as 48 bytes data + 5 byte header information. The header information includes source machine name, destination machine name, position of the packet and the related information. The message as packets is sent to the router where it splits and gets the destination address name and forwards the message's packet to the destination. The destination also splits the packet information and then extracts the original message from the packets and sorts it using the hash algorithm based on the index or position.

D. Timer control

Whenever each and individual packet starts sending a timer is started. The system current time is taken as a start time and added with delay and it acts as a threshold time and if the threshold time exceeds the maximum elapsed time of the packet then the packet is retransmitted. If the time doesn't exceed then the packet may arrive safe. If so the next packet is transmitted else the current packet is transmitted until it arrives safely. Thread concept is used to implement the timer.

E. Packet Reordering C.

- Collects packets from intermediate routers.
- Reorders the packets using hash table.
- When EOF is reached it sends the message to destination.

F. Comparison Chart

Comparison chart compares the throughput of TCP without Packet Reordering with New TCP With packet Reordering. The performance is shown by comparing the transmission rate of existing system with proposed system

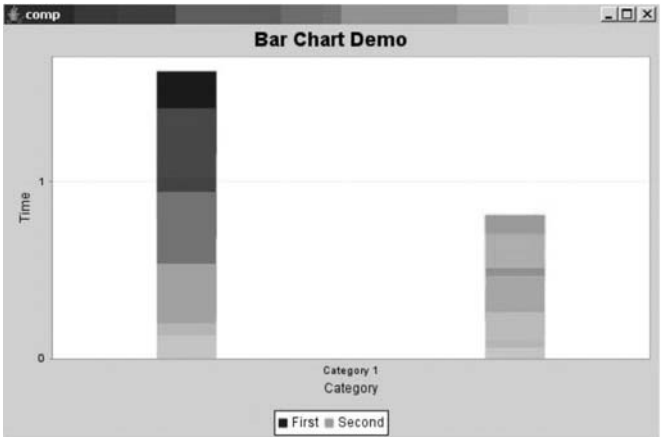

Figure I. comparing the transmission rate of existing system with proposed system

I. TRANSMISSION CONTROL PROTOCOL-CONSTANT PACKET REORDERING

The basic idea behind TCP-CPR is to detect packet losses through the use of timers instead of duplicate acknowledgments. This is prompted by the observation that, under constant packet reordering, duplicate acknowledgments are a poor indication of packet losses. Because TCP-CPR relies solely on timers to detect packet loss, it is also robust to acknowledgment losses as the algorithm does not distinguish between data (on the forward path) or acknowledgment (on the reverse path) losses.

The proposed algorithms only require changes in the TCP sender and are therefore Backward-compatible with any TCP receiver. TCP-CPR's sender algorithm is still based on the concept of a congestion window, but the update of the congestion window follows slightly different rules than standard TCP. However, significant care was placed in making the algorithm fair with respect to other versions of TCP to ensure they can coexist

II. ALGORITHMS

Packets being processed by the sender are kept in one of two lists: the to-be-sent list contains all packets whose transmission is pending, waiting for an "opening" in the congestion window. The to-be-ack list contains those packets that were already sent but have not yet been acknowledged. Typically, when an application produces a packet it is ûrst placed in the to-be-sent list; when the congestion window allows it, the packet is sent to the receiver and moved to the to-be-ack list; ûnally when an ACK for that packet arrives from the receiver, it is removed from the to-be-ack list (under cumulative ACKs, many packets will be simultaneously removed from to-be-ack). Alternatively, when it is detected that a packet was dropped, it is moved from the to-be-ack list back into the to-be-sent list.

As mentioned above, drops are always detected through timers. To this effect, whenever a packet is sent to the receiver and placed in the to-be-ack list, a timestamp is saved. When a packet remains in the to-be-ack list more than a certain amount of time it is assumed dropped. In particular, it assumed that a packet was dropped at time when exceeds the packet's timestamp in the to-be-ack list plus an estimated maximum possible round-trip time mxrtt.

As data packets are sent and ACKs received, the estimate mxrtt of the maximum possible round-trip time is continuously updated. The estimate used is given by

$$\texttt{mxrtt} := \beta \times \texttt{srtt} \qquad (1)$$

Where â is a constant larger than 1 and srtt an exponentially weighted average of past RTT Whenever a new ACK arrives, srtt updated as follows:

$$\texttt{srtt} = \max\left\{\alpha^{\frac{1}{\lfloor \texttt{cwnd} \rfloor}} \times \texttt{srtt}, \texttt{sample} - \texttt{rtt}\right\} \qquad (2)$$

Where á denotes a positive constant smaller than 1, $\lfloor \texttt{cwnd} \rfloor$ the ûoor of the current congestion window size, and sample - RTT the RTT for the packet whose acknowledgment just arrived. The reason to raise á to the power 1/ is that in one RTT the formula in (2) is iterated times. This means that, e.g., if there were a sudden decrease in the RTT then srtt would decrease by a rate of per RTT, independently of the current value of the congestion window. The parameter á can therefore be interpreted as a smoothing factor in units of RTT.The performance of the algorithm is actually not very sensitive to changes in the parameters â and á, provided they are chosen in appropriate ranges.

Note that srtt tracks the peaks of RTT. The rate that srtt decays after a peak is controlled by á. The right-hand plot shows how large jumps can cause RTT > mxrtt (for this data set, occurrences at 15 s, 45 s, 75 s, etc.) resulting in spurious time- outs (note that the jumps in RTT in the right-hand plot were artiûcially generated). In order for these jumps to cause a spurious timeouts, the jumps in RTT could occur no sooner than every 15 seconds. In this case, 1500 packets were delivered between these jumps. If the jumps occurred more frequently, then, as can be seen from the ûgure mxrtt, would not have decayed to a small enough value and spurious timeouts would not occur. Furthermore, if the jumps were larger, then the time between jumps to cause a timeout would be no smaller.

Two modes exist for the update of the congestion window: slow-start and congestion-avoidance. The sender always starts in slow-start and will only go back to slow-start after periods of extreme losses. In slow-start, cwnd starts at 1 and increases exponentially (increases by one for each ACK received). Once the ûrst loss is detected, cwnd is halved and the sender transitions to congestion-avoidance, where cwnd increases linearly (1/ cwnd for each ACK received). Subsequent drops cause further halving of cwnd, without the sender ever leaving

congestion-avoidance. An important but subtle point in halving cwnd is that when a packet is sent, not only a timestamp but the current value of cwnd is saved in the to-be-ack list. When a packet drop is detected, then cwnd is actually set equal to half the value of cwnd at the time the packet was sent and not half the current value of cwnd. This makes the algorithm fairly insensitive to the delay between the times a drop occurs until it is detected.

To prevent bursts of drops from causing excessive decreases in cwnd, once a drop is detected a snapshot of the to-be-sent list is taken and saved into an auxiliary list called memorize. As packets are acknowledged or declared as dropped, they are removed from the memorize list so that this list contains only those packets that were sent before cwnd was halved and have not yet been unaccounted for. When a packet in this list is declared dropped, it does not cause cwnd to be halved. The rational for this is that the sender already reacted to the congestion that caused that burst of drops. This type of reasoning is also present in TCP-New Reno and TCP-SACK. The pseudo-code in Table I corresponds to the algorithm just described

Event	Code
Initialization	1 mode = slow-start 2 cwnd = 1 3 ssthr = +8 4 memorize = θ
Time>time(n) + mxrtt (drop detected for packet n)	5 remove(to-be-ack, n) 6 add(to-be-sent, n) 7 if not is-in(memorize , n) then /*new drop*/ 8 memorize = to-be-ack 9 cwnd = cwnd(n)/2 10 ssthr =cwnd 11 else /*other drop is burst*/ 12 remove(memorize , n) 13 flush-cwnd{}
Ack received for packet n	14 srtt = max{α 1/cwnd x srtt, time-time(n)} 15 mxrtt = β x srtt 16 remove(to-be-ack, n) 17 remove(memorize, n) 18 if mode= slow-start and cwnd +1= ssthr then 19 cwnd = cwnd + 1 20 Else 21 Mode = congestion- avoidance 22 cwnd = cwnd + 1/ cwnd 23 flush-cwnd{}
flush-cwnd{}	24 while cwnd>| to-be-ack| do 25 k=sent(to-be-sent) 26 remove(to-be-sent,k) 27 add(to-be-ack, k) 28 time(k) = time

TABLE I Table I .shows the Algorithm for TCP-PR

I. IMPLEMENTATION

Implementation is the stage in the work where the theoretical design is turned into a working system and is giving confidence on the new system for the users, which it will work efficiently and effectively. It involves careful planning, investigation of the current System and its constraints on implementation, design of methods to achieve the change over, an evaluation, of change over methods. Apart from planning major task of preparing the implementation are education and training of users. The more complex system being implemented, the more involved will be the system analysis and the design effort required just for implementation.

An implementation co-ordination committee based on policies of individual organization has been appointed. The implementation process begins with preparing a plan for the implementation of the system. According to this plan, the activities are to be carried out, discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system.

Implementation is the final and important phase, the most critical stage in achieving a successful new system and in giving the users confidence. That the new system will work be effective .The system can be implemented only after through testing is done and if it found to working according to the specification. This method also offers the greatest security since the old system can take over if the errors are found or inability to handle certain type of transactions while using the new system.

II. CONCLUSIONS

In this paper proposed and evaluated the performance of Transmission control protocol constant Packet Reordering (TCP-CPR), a variant of Transmission control protocol(TCP) that is speciûcally designed to handle constant reordering of packets (both data and acknowledgment packets). Our simulation results show that TCP-CPR is able to achieve high throughput when packets are reordered and yet is fair to standard TCP implementations, exhibiting similar performance when packets are delivered in order. From a computational view-point, TCP-CPR is more demanding than TCP but carries essentially the same overhead as Selective Acknowledgement Options (TCP-SACK).

Because of its robustness to constant packet reordering, TCP-CPR allows mechanisms that introduce constant packet reordering as part of their normal operation to be deployed in

the Internet. Such mechanisms include proposed enhancements to the original Internet architecture such as multi-path routing for increased throughput, load balancing, and security; protocols that provide differentiated services and traffic engineering approaches.

REFERENCES:

[1] Bohacek, S.; Hespanha, J.P.; Junsoo Lee; Lim, C.; Obraczka, K. "A new TCP for persistent packet reordering" Networking, IEEE/ ACM Transactions on
Volume 14, Issue 2, April 2006 Page(s): 369 – 382

[2] Colin M. Arthur, Andrew Lehane, David Harle, "Keeping Order: Determining the Effect of TCP Packet Reordering," icns, pp.116, International Conference on Networking and Services (ICNS '07), 2007

[3] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," ACM Comput. Commun. Rev., vol. 32, no. 1, 2002.

[4] J. Bennett and C. Partridge, "Packet reordering is not pathological network behavior," IEEE/ACM Trans. Netw., vol. 7, no. 6, pp. 789–798,Dec. 1999.

[5] F. Wang and Y. Zhang, "Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response," in ACM MOBIHOC, 2002, pp. 217–225.

[6] T. Dyer and R. Boppana, "A comparison of TCP performance over three routing protocols for mobile ad hoc networks," in ACM MOBIHOC,2001, pp. 56–66.

[7] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad-hoc networks," in ACM MOBICOM, 1999, pp. 219–230.

[8] N. Taft-Plotkin, B. Bellur, and R. Ogier, "Quality-of-service routing using maximally disjoint paths," in Proc. IEEE/IFIP IWQoS'99, Jun.1999, pp. 119–128.

[9] S. Blake, D. Black, M. Carlson, E. Davies, Z. Whang, and W. Weiss,"An architecture for differentiated services," RFC 2475, 2005.

[10] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An extension to the Selective Acknowledgment (SACK) option for TCP," RFC 2883,2000.

[11] D. Bertsekas, Network Optimization: Continuous and Discrete Models. Belmont, MA: Athena Scientific, 2006.

[12] M. Allman and V. Paxson, "Computing TCP's retransmission timer,"RFC 2988, Nov. 2000.

[13] N. Zhang, B. Karp, S. Floyd, and L. Peterson, RR-TCP: A reorderingrobust TCPWith DSACK. ICSI, Berkeley, CA, Tech. Rep. TR-02-006, Jul. 2002

[14] S. Bohacek, J. Hespanha, K. Obraczka, J. Lee, and C. Lim, "Secure stochastic routing," presented at the ICCCN'02, Miami, FL, 2002.

[15] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker, "Characterizing and measuring path diversity of Internet topologies," presented at the ACM SIGMETRICS, San Diego, CA, Jun. 2003.

[16] A. Nasipuri and S. Das, "Demand multipath routing for mobile ad hoc networks," presented at the ICCCN'99, Boston, MA, Oct. 1999.

[17] M. Pearlman, Z. Haas, P. Sholander, and S. Tabrizi, "The impact of alternate path routing for load balancing in mobile ad hoc networks,"presented at the ACM MobiHoc, Boston, MA, Aug. 2000.

[18] R. Ludwig and R. Katz, "The Eifel algorithm: making TCP robust against spurious retransmissions," ACM Comput. Commun. Rev., vol.30, no. 1, 2000.

[19] S. Bhandarkar, N. Sadry, A. L. N. Reddy, and N. Vaidya, "TCP-DCR: a novel protocol for tolerating wireless channel errors," IEEE Trans.Mobile Comput., vol. 4, no. 5, pp. 517–529, Sep.-Oct. 2004.

[20] B. Sikdar, S. Kalyanaraman, and K. S.Vastola, "Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno, and SACK,"IEEE/ACM Trans. Netw., vol. 11, no. 6, pp. 959–971, Dec. 2003.

[21] S. Bohacek, "A stochastic model of TCP and fair video transmission," in Proc. IEEE INFOCOM, 2003, pp. 1134–1144.