International Journal of Computer Applications (0975 – 8887)
Volume 51– No.9, August 2012

Dynamically Avoiding the Substantial Throughput Penalty of FRR

Ramesh K
Dept. Of P.G. Studies in Computer Science
Karnatak University Dharwad,
India

Zameer Ahmed
Dept. Of P.G. Studies in Computer Science
Karnatak University Dharwad,
India

ABSTRACT

Disclosed is a system for dynamically suppressing Fast Retransmission and Recovery (FRR) on high-latency Transport Control Protocol (TCP) connections with low ongoing packet loss. This system measures the relative rate of recent packet loss and adjusts the suppression of FRR based on the measured rate. Provided that the rate of actual packet loss is low, high-latency TCP connections can benefit significantly from suppressing FRR.

Keywords

Latency, Bandwidth, Throughput, Round-Trip Latency, TCP, FRR, SACK

1. INTRODUCTION

The TCP protocol incorporates flow control mechanisms to attempt the best utilization of network bandwidth. The most common mechanism in use is documented by RFC 2581 and is sometimes called TCP Reno. Most TCP implementations severely penalize a connection's throughput in case of packet loss, and regain that throughput as stable packet transmission occurs. This represents a particular problem for high-latency, high-bandwidth networks (so-called "long fat pipes"), which are increasingly common. Because of the high bandwidth of such networks, a severe cut in connection throughput represents a substantial loss of throughput, even for reasonably small rates of packet loss. And because of the high latency in such networks, it can take quite some time for the connection to restore its bandwidth, since assurance of successful packet transmission is delayed by the network latency.

1. 1 Network Throughput

Network throughput is the average rate of successful message delivery over a communication channel. This data may be delivered over a physical or logical link, or pass through a certain network node. The throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second or data packets per time slot. The system throughput or aggregate throughput is the sum of the data rates that are delivered to all terminals in a network

1.2 Bandwidth

Bandwidth is technically defined as the amount of information that can flow through a network at a given period of time. This is, however, theoretical- the actual bandwidth available to a certain device on the network is actually referred to as throughput.

1.3 Round-trip latency

To solve the universal clock problem, some test equipment has the ability to sync up with a GPS clock and place a timestamp inside the packet that is sent to measure latency. The receiving device is a similar piece of equipment that can also sync up with a GPS clock. This device then compares the time that the packet is received with the timestamp in the received packet to obtain an end-to-end latency measurement for that packet. This option is very expensive. Fortunately, there is a cost efficient method that provides acceptable accuracy. If the sender to the receiver path is the same as the path from the receiver to the sender, the round-trip latency (latency from the sender to the receiver and back) can be measured and assumed that the end-to-end latency is half of this result. Measuring round-trip latency is easy and details of this will be covered later in this document. Measuring round-trip latency means that all time comparisons are made from the same device, which removes the need for devices to sync to a common clock. It also solves the problem of keeping up with the send and receives times for each packet since these times are all associated with one packet in one device.

2. TCP BEHAVIOUR

2.1 Fast Retransmission and Recovery

Fast Retransmit is an enhancement to TCP which reduces the time a sender waits before retransmitting a lost segment. A TCP sender uses a timer to recognize lost segments. If an acknowledgement is not received for a particular segment within a specified time (a function of the estimated Round-trip delay time), the sender will assume the segment was lost in the network, and will retransmit the segment. Duplicate acknowledgement is the basis for fast retransmit mechanism which works as follows: after receiving a packet (i.e., sequence number 1), the receiver sends an acknowledgement adding 1 with the sequence number (i.e., sequence number 2) which means that the receiver receives the packet number 1 and it expects packet number 2 from the sender. Let’s assume that three subsequent packets have been lost. In the meantime the receiver receives the packet number 5 and 6. After receiving packet number 5, the receiver sends an acknowledgement with the sequence number 2 and 6. When the receiver receives packet number 6, it sends acknowledgement with the sequence number 2 and 7. In this way, the sender receives more than one acknowledgement with the same sequence number 2 which is called duplicate acknowledgement. The fast retransmit enhancement works as follows: if a TCP sender receives a specified number of acknowledgements which is usually set to three duplicate acknowledgements with the same acknowledge number (that is, a total of four acknowledgements with the same acknowledgement number), the sender can be reasonably confident that the segment with the next higher sequence number was dropped, and will not arrive out of order. The sender will then retransmit the packet that was presumed dropped before waiting for its timeout.

2.2 TCP Selective Acknowledgment Options

Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput. TCP uses a cumulative acknowledgment scheme in which received segments that are not at the left edge of the receive window are not acknowledged. This forces the sender to either wait a roundtrip time to find out about each lost packet, or to unnecessarily retransmit segments which have been correctly received. With the cumulative acknowledgment scheme, multiple dropped segments generally cause TCP to lose its ACK-based clock, reducing overall throughput. Selective Acknowledgment (SACK) is a strategy which corrects this behavior in the face of multiple dropped segments. 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. The selective acknowledgment extension uses two TCP options. The first is an enabling option, "SACK-permitted", which may be sent in a SYN segment to indicate that the SACK option can be used once the connection is established. The other is the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted. The SACK option is to be included in a segment sent from a TCP that is receiving data to the TCP that is sending that data; we will refer to these TCPs as the data receiver and the data sender, respectively. We will consider a particular simplex data flow; any data flowing in the reverse direction over the same connection can be treated independently.

3. TO DISABLE TCP/IP TIMESTAMPS

The Problem: TCP timestamps are enabled on the remote host. This could allow a remote attacker to estimate the amount of time since the remote host was last booted.

3.1 Resolution:

TCP timestamps are generally only useful for testing, and support for them should be disabled if not needed.

3.2 To disable TCP timestamps on Linux

Add the following line to the /etc/sysctl.conf file:net.ipv4.tcp_timestamps = 0.

3.3 To disable TCP timestamps on Windows

Set the following registry value:
Key:
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\ParametersValue: Tcp1323Opts. Data: 0 or 1.

3.4 To disable TCP timestamps on Cisco

Use the following command:
no ip tcp timestamp.

4. PREVIOUS WORK

There are numerous existing schemes to correct this problem and ensure that TCP connection throughput more closely parallels the actual capacity of the network. Some of these schemes are as follows:

1. Ravot proposes that the rate of throughput growth after packet loss be increased by a greater factor than that allowed by RFC 2581.
2. Mascolo et. al. propose that statistical measures of packet throughput be used to estimate actual network capacity, and determine a reasonable TCP connection throughput; this approach is known as TCP Westwood.

Experimental RFC 2861 proposes adjustments to RFC 2581 that ensure that (1) TCP flow control does not allow excessive packet bursts after periods of inactivity; and (2) TCP throughput is penalized less severely in case of packet loss. Experimental RFC 3649 proposes similar improvements to the penalization of connections upon packet loss. Bandwidth reservation may be employed using Resource Reservation Protocol (RSVP) or underlying network services such as Asynchronous Transfer Mode (ATM) to assure a specific TCP connection throughput that can be attained without the need to estimate network capacity. BIC TCP (Binary Increase Congestion control for TCP) is used by some platforms (e.g., Linux kernels 2.6.8 and above) to improve the rate of throughput increase after packet loss and throughput penalization. Binary search is used to more rapidly approach and discover the optimal throughput rate the network can tolerate. FAST TCP attempts to statistically measure queuing capacity in the network and use this to estimate the optimal throughput for TCP connections. While RFC 2581 is optimized for the case where individual packet loss occurs, RFC 2018 documents a "selective acknowledgement" (SACK) mechanism whereby TCP can more efficiently recover from multiple packet loss.

4.1 Problem: Above Said

Many of these solutions are independent and can be employed in combination with one another. Many of these solutions seek to reduce the impact of penalizing throughput on packet loss. By contrast, the system proposed herein attempts to suppress that penalty altogether in cases where packet loss may not be occurring. Once sufficient packet loss occurs, other methods such as those cited above may be used to reduce the impact of throughput penalization. For pages other than the first page, start at the top of the page, and continue in double-column format. The two columns on the last page should be as close to equal length as possible.

5. PREVIOUS WORK

The system and method proposed herein involves suppressing the Fast Retransmission and Recovery (FRR) feature of RFC 2581 on high-latency TCP connections until packet loss occurs and retransmission becomes necessary. This technique relies on the fact that high-latency high-throughput connections are particularly susceptible to packet reordering, which can give the appearance of packet loss but which should not cause a connection's throughput to be penalized. However, when genuine packet loss begins to occur on such connections, FRR should be enabled to recover from the loss efficiently. This system dynamically measures the rate at which real packet loss is occurring on a high latency TCP connection, compared to the rate at which packet reordering is occurring on that connection. At any given time, when the recent rate of apparent packet loss due to packet reordering exceeds the recent rate of genuine packet loss, this system will suppress FRR for a high-latency connection, recognizing that the apparent packet loss is likely not real, and the connection should not be subjected to an FRR throughput penalty.

5.1 Working Mechanism

This system uses a side effect of TCP timestamps to discover the case where packet reordering is causing only apparent packet loss. The system maintains a counter for each TCP connection, dynRetrans that is initialized to zero when the connection is established. This counter is incremented (but not greater than a ceiling value, such as 10) whenever TCP retransmission occurs (either due to retransmission timeout or due to FRR), representing the fact that packet loss occurred. If TCP timestamps are enabled, this counter is decremented by two (but not less than zero) whenever: A packet is retransmitted, an acknowledgment is received for that packet. The TCP timestamp echoed in the acknowledgment is less than the TCP timestamp for the retransmitted packet. This logic identifies cases where the acknowledgment pertains to the original packet and not the retransmitted packet. Thus, the dynRetrans counter is decremented whenever retransmission occurred but it proved unnecessary because the original packet was not lost. (Since the counter is incremented whenever retransmission occurs, we decrement by two so that any unnecessary retransmission results in a net decrement of one). The dynRetrans counter is thus a dynamic measure of whether ongoing packet loss is occurring. As packet loss occurs and real retransmissions take place, it will grow. As retransmissions prove unnecessary (the original packet actually arrived at the destination), it will shrink. The TCP connection then uses this counter to identify whether to suppress FRR for a high-latency connection. If the counter is less than or equal to a certain value (e.g., 3), the connection is considered to have low real packet loss, and FRR is suppressed for this connection until the rate of packet loss measured by dynRetrans exceeds that value. The connection thus dynamically avoids the substantial throughput penalty of FRR for any time period during which packet loss is insignificant.

6. MECHANISM

6.1 Algorithm

```
---------------------------------------------------------------------------
Step 1 :   Set dynRetrans = 0
Step 2 :  If Timestamp is invalid then
          For each Retransmission Occurred
          dynRetrans = dynRetrans + 1
          Until dynRetrans = 10
          End for
          Else
           For each retransmission occurred and ack is received
           If timestamp(ack) < timestamp (
           retransmitted packet) then
           dynRetrans = dynRetrans – 2
           End if
           Until dynRetrans > = 3
           End For
           End if
Step 3 :  If dynRetrans < = 3
           Stop Retransmission
           Else
           GoTo Step 2
---------------------------------------------------------------------------
```

6.2 Implementation

```
-------------------------------------------------------------------
main()
{
int dynRetrans=0,timestamp=0,retrans,ack;
int retranspack;
            if(timestamp=0)
            {
                        if(retrans=1)
                        {
                                    while(dynRetrans<=10)
                                    dynRetrans+=dynRetrans;
                        }
}
else
{
if(retrans=1 && ack=1)
                        {
                        if(ack_timestamp<ack_retrans_pack)
                                    {
                                                while(dynRetrans<=3)
{

            dynRetrans=dynRetrans-2;
                                                }
                                    }
                        }
}
if(dynRetrans<=3)
   retrans=0;
}
-------------------------------------------------------------------
```

6.3 Subroutine to calculate Throughput

```
-------------------------------------------------------------------
int throughput_with_timestamp( )
{
            int time = 10;
            while (time< = 100)
            {
                        throughput_timestamp =
tcp_window_size/latency/1000000 * time;
                        time = time + 10;
            }
            return(throughput);
}
int throughput_notmsstmp( )
{
             int time = 10;
            while(time<=100)
            {
throughput_notmsstmp =
tcp_window_size/latency/1000000*time*2;
                        time=time+10;
            }
            return(throughput);
}
-------------------------------------------------------------------
```

7. RESULTS AND COMPARISONS

Table 1. Throughput with and without timestamp

Time (mSec)	Throughput with timestamp (Mbps)	Throughput without timestamp (Mbps)
10	87.381	174.76
20	174.763	349353
30	262.144	524.29
40	349.525	699.05
50	436.907	873.81
60	524.288	1048.58
70	611.669	1223.34
80	699.051	1398.10
90	786.432	1572.86
100	873.813	1747.63

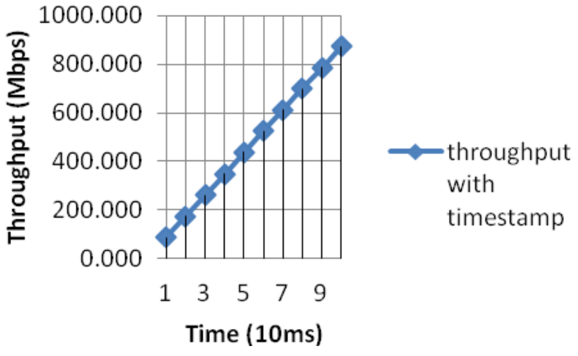

Fig 1: Throughput with timestamp

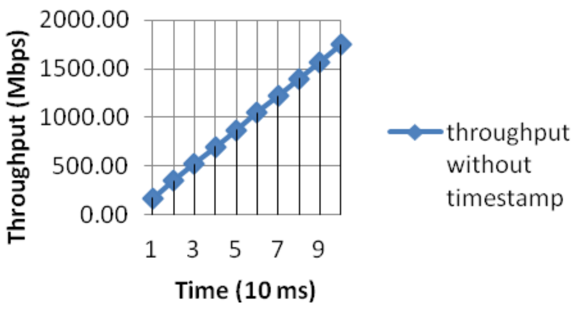

Fig 2: Throughput without timestamp

8. CONCLUSION

Dynamical suppression of FRR for TCP implementations will avoid the severe penalize of connection's throughput in case of packet loss, this mechanism will regain the throughput as stable packet transmission occurs. This solves a particular problem for high-latency, high-bandwidth networks, which are increasingly common. Because of the high bandwidth of such networks, a severe cut in connection throughput represents a substantial loss of throughput, even for reasonably smaller rate of packet loss. And because of the high latency in such networks, it can take quite some time for the connection to restore its bandwidth, since assurance of successful packet transmission is delayed by the network latency. Study of TCP behavior is still an active area of research and requires further investigation since not much of the work is done beside the ones described in this article.

9. REFERENCES

[1] "Results on High Throughput and Quos Between the US and CERN." HENP Special Interest Group. Retrieved September 4, 2007, from http://henp.internet2.edu/Ravot.ppt.

[2] Mascolo, Saverio et. al. "TCP WESTWOOD Home Page." UCLA Computer Science Department. Retrieved September 4, 2007, from http://www.cs.ucla.edu/NRL/hpi/tcpw/.

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

[4] Van Jacobson. Congestion Avoidance and Control. In proceedings of ACM SIGCOMM, 1988.

[5] V. Jacobson, Modified TCP Congestion Avoidance Algorithm, end2end internet mailing list.

[6] W.Stevens, "TCP Slow Start, Congestion Avoidance Algorithm, Fast Retransmit and Fast Recovery Algorithms", RFC 2001. Jan 1997.

[7] M. Allman, V. Paxson, W. Stevens, " TCP Congestion Control", RFC 2581, Apr.1999

[8] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version), IEEE/ACM Transactions on Networking, Vol 2, Feb. 1994

[9] B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenkar, J. Wroc lawski, L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the internet", RFC 2309, April. 1998.

[10] Sally Floyd and Van Jacobson, Random Early Detection Gateways for congestion Avoidance", IEEE/ACM Transaction on Networking, Aug. 1993.

[11] Eman Hashem. Analysis of random gateway congestion control. Master's thesis, Massauchusetts Institute of Technology, 1989. MTL/LCS/TR-465.

[12] Scott Shankar, Lixia Zhang and David Clark. Some observations on the dynamics of a congestion control algorithm. In proceedings of a ACM SIGCOMM, 1990.

[13] Sally Floyd and Van Jacobson. On traffic phase effects in packet-switched gateways. Internetworking: Research and Experience, 3(3), September 1992.

[14] Allison Mankin. Random drop congestion control. In proceedings of ACM SIGCOMM, 1990.

[15] John Nagle, RFC 896 : Congestion control in IP/TCP internetworks. Technical Report, internet Assigned numbers Authority, Jon Postel, USC/ISI, 4676 Admoralty Way, Marina Del Rey, DA 90292, 1984. http://info.internet.isi.edu/in-notes/rfc/files/rfc896.txt.

[16] Raj Jain. A timesout-based congestion control scheme for window flow-controlled networks. IEEE Journal on Selected Areas in Communications, SAC-4(7):1162-1167, October 1986.

[17] Phil Karn and Craig Partridge. Improving round-trip time estimates in reliable transport protocols. In proceedings of ACM SIGCOMM, 1993.

[18] Mark Handley. An examination of mbone performance. Technical report, University of southern California Information sciences Institute, 1997. ISI/RR-97-450.