TCP Congestion Control

Onwutalobi, Anthony Claret
Department of Computer Science
University of Helsinki,
Helsinki Finland
onwutalo@cs.helsinki.fi

Abstract— **This paper is aimed to discuss congestion control and review TCP functionality that provides a congestion control techniques of handling buffer –overflows, loss of packets and congestions that we experience over a network. The primary focus is on discussing the four intertwined algorithms, namely slow start, congestion avoidance, fast retransmit, and fast recovery, which will help us to solve this congestion problem. The algorithms are rooted in the idea of achieving network stability. We demonstrate in this paper how the implementation of this algorithm reduces congestion over congested network and more importantly how it scales gracefully in the network environment by providing high utilization and low queuing delays.**

***Index Terms*—congestion avoidance, congestion window, fast retransmit, fast recovery, slow start, TCP**

I. INTRODUCTION

In the 1920's William L. Potts [1] faced with the problem of persistent heavy traffic in Detroit Michigan, invented a four-way three traffic light system. This invention was primary to address the issue of increasing influx of automobiles. Potts used a simple algorithm of color-selection signal where Red (Road is busy), green (Road is free), and amber (warning road is getting congested) colors are use to regulate the traffic congestion.

In the computer network environment, congestion problem is analogous to heavy traffic problem experienced during Potts's time. Given that in these recent times, Internet and computer networks are experiencing an explosive growth. Most daily transactions and activities are done via the network in the form of sharing of files, transmitting of data or computing resources. This increasing growth of computer network thus has posed severe congestion problems. For example, it is very common now to hear that internet gateways are tremendously falling down. Unfortunately these gateways (routers) do not have robust storage capacity, and situation deteriorates most times when congestion occurs, whereby the datagrams arriving at the congested router grows until the router reaches its storage limit and begins to drop datagrams. The most delicate aspect of this situation is that most times, the two endpoints (sender and receiver), when confronted with this kind of problem, do not usually know the details of what the cause of the congestion is and where and why it occurred. We know that TCP uses timeout and retransmission technique in responding to congestion (data loss). Therefore, the endpoints normally assume that the timeout is due to simple delay in the network. In response to resolve congestion, they retransmit the presumed loss datagram by sending more datagrams, which however worsen the congestion instead of solving the problem. The whole process, if not limited, renders the network useless as more and more datagrams are been retransmitted in an attempt to alleviate the congestion problem. This situation often results to what is known as congestion collapse.

In this seminar paper, we present the algorithm that can solve this problem, in order to avoid this chaos of congestion collapse. We will discuss the implementation of TCP standard using the following techniques: slow-start and congestion avoidance [2].Basically, the two techniques primarily reduce the rate of datagrams injected in the network but do not address the issue of network throughput and efficiency. In order to maintain the robustness of the network and achieving network utilization and throughput, this paper will also give a comprehensive treatment on some approaches to handle network "throughput" and "efficiency" by introducing fast recovery and fast retransmit techniques. We will show how the algorithms are derived from this principle and what effect they have on traffic over congested networks. The scope of our discussion is limited to the stated algorithm to enhance network stability by forcing the transport connection to follow the techniques. References will be made from different sources and RFC Documents as a primary source [3] .

II. BACKGROUND - TCP PROTOCOL

A. *User Datagram Protocol*

In the TCP/IP protocol suite, UDP provides primary mechanism that application programs use to send datagrams to

other application programs [2]. It uses the underlying Internet layer to transport a message from one machine to another, and provide unreliable, connectionless datagram delivery. Packets can be lost or destroyed when network hardware fails, or when network becomes heavily loaded. [2].In addition, understanding that UDP is connectionless, most application packets sent, may arrive out of order or sometimes delivered after a long delay. Knowing fully well that most application program is sent in a large volume, it becomes a tedious task for programmers to write programs that will solve the reliability problem. Although it is obvious that such task may not be successful. It has then become necessary to develop general purpose program or algorithm that will ensure reliable stream delivery and also address the issue of data loss and damage on transit. This new protocol will provide platform independent and also cover wider-range of different application programs while hiding the details of networking to enable a uniform interface for the stream transfer services [4]

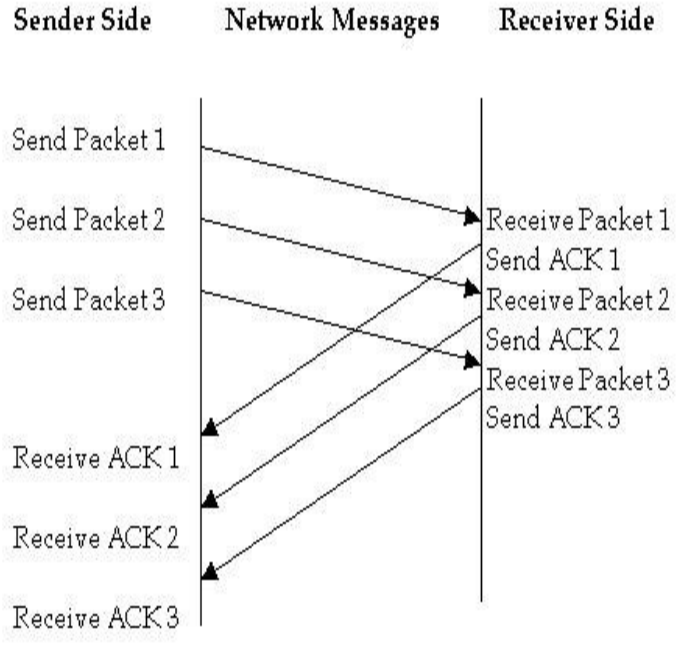
According to [2], the properties of reliable delivery services were discussed in details which guarantees in principle, a reliable delivery with no duplicate or data loss as opposed to UDP. The success story behind this TCP technique is known as positive acknowledgement with retransmission. This mechanism requires both endpoints to be on a constant communication whereby the sender sends application packages (datagram) and wait for an acknowledgement from the receiver. The sender stores the history of the packages sent and also set a timer when it sends a package. It waits for a reply for a set periodic and will retransmit the package if no reply ACK received.

This protocol solves the problem of connectionless UDP protocol by ensuring the reliability of data transmitted over the network. However it is too mechanical and it wastes a lot of network bandwidth. It also has poor utilization of network resources because the sender must delay sending a new package until it receives an acknowledgement for the previous packet [2].

This limitation propels a research and introduction of sliding window. This sliding window protocol keeps the network completely saturated with the packets. It provides a reasonable throughput than a simple positive acknowledgement protocol. A sliding window protocol keeps track of transmitted and acknowledged packets. Since the timer is set for each datagram sent over the network, the lost packets are easily identified and retransmitted once their timer had expired. This sliding window protocol is connection oriented which guarantees data connection. It also born several techniques that help in the enhancement of effective and efficient data flows in the network.

In the diagram below, is an example on how Positive acknowledgement protocol transfer data. It transmits three packets using a sliding window protocol. The key concept is the sender can transmit all packets in the window without waiting for an acknowledgement. [5]

Figure 1: Positive Acknowledgement Protocol Transfer



III. CONGESTION CONTROL

According to [2], congestion is a condition of severe delay caused by an overload of datagrams at one or more switching point (router). On the other hand, congestion control is a distributed algorithm to share network resources among competing users. [7] This control is been carried out by the Transmission Control Protocol (TCP)

In the past, TCP started its connection by allowing the sender to send maximum number of segments advertised by the receiver without considering if there are slower links between the sender and the receiver or if the router has enough space capacity to handle the multiple injections of packets. This earlier version of TCP was good for two endpoints that exchange data on the same network. Problems can set in immediately either endpoint is in a different network. Another problem can occur if the router storage capacity is not enough to hold the packages sent. Due to this limitation, there was a need to create a congestion control algorithm that can handle these problems. According to RFC [3] the most eligible technique to tackle this problem is using an algorithm called slow start [2]

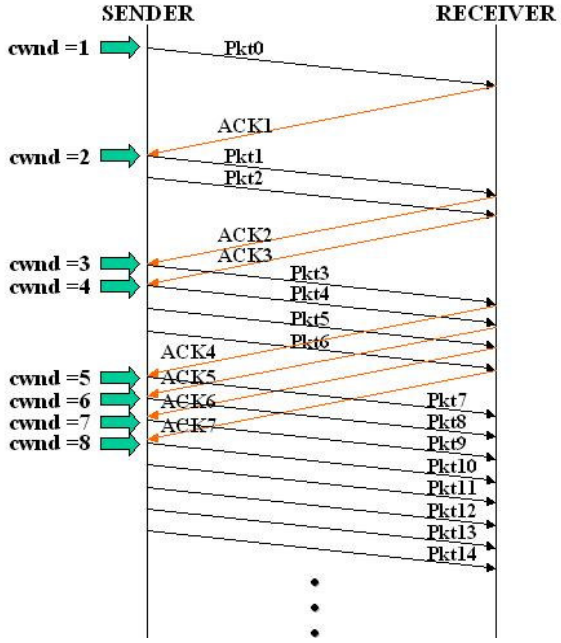
IV. SLOW START

The slow start algorithm regulates the flow of datagrams in the network to avoid congestion. With slow start algorithm, TCP monitors to make sure that the rate new packets are sent over the network are the rate at which the acknowledgements are returned by the receiver. Slow start is normally used when starting traffic on a new connection or when recovering from congestion. In this process, extra datagram can only be sent through the network only when there is a receipt of acknowledgement from the receiver. The slow-start window is intended to be incremented exponentially. Below is the algorithm of slow start from [4]

Add a congestion window, cwnd, to the per-connection state.
When starting or restarting after a loss, set cwnd to one packet
On each Ack for new data, increase cwnd by one packet.
When sending, send the minimum of the receiver's advertised window and cwnd. [3]

With this algorithm, TCP regulates the data flows and tries to maintain a congestion window by limiting the total number of unacknowledged packets that may be in transit. Slow start avoids swamping the entire network with extra traffic immediately after congestion clears or when new connections are started. The sender initializes the congestion window to 1, and sends a datagram to the other end and waits to receive an acknowledgement before another packet will be sent. On acknowledging that the receiver has received the datagram through the Ack sent by the receiver, the sender sends 2 more packets; one for the Ack and one because an Ack opens the congestion window by one packet and wait for Ack. When the two acknowledgments arrive the two endpoint increase the congestion window by 2, So TCP can send 4 segments. Acknowledgment of those will increase the congestion window to 8 and so on. This is what we mean by exponential increase. Actually, the slow start window increase is not fully exponential: it takes time Rlog2 W where R is the round trip-time and W is the window size in packet [9] what this means is that the rate window open is very fast that it does not have any serious effect in performance even on the links with a large bandwidth-delay product [9]

Figure 2:



The diagram below shows the scenario of TCP slow-start mechanism:
Assumption: The maximum congestion window size is 8. Therefore, the congestion window size will not increase after it reaches the size 8. The sender starts with the congestion window size 1. Upon each ACK from the receiver, the sender increases the congestion window size by 1. For example, when the sender receives "ACK 1", its congestion window size is increased to be 2. Thus, in the next window, the sender can send 2 packets consecutively without receiving any ACKs from the receiver.

To avoid increasing the window size too quickly and causing additional congestion, TCP also imposes an additional restriction. TCP checks the congestion window of the sender, once it has reached one half of its initial size before congestion or has exceeded a threshold ssthresh, or a packet is lost, it enters a congestion avoidance phase and reduces the rate of increment. According to [8], slow start usually ends after a loss since the initial ssthresh is large. However, ssthresh is updated at the end of each slow start, and will often affect subsequent slow starts triggered by timeouts .

V. CONGESTION AVOIDANCE.

As mentioned in [2] congestion avoidance is a TCP restriction technique of regulating slow start exponential duplication method to avoid flooding the network with segment which could cause congestion. The fastest method of determining when congestion avoidance should be implemented is once the

receiver receives Ack from the sender over a lost packet. Since TCP sets a timer for every packet sent, if the timers are in good condition, it is possible to state with confidence that the timeout that occurred is not a broken timer but indicates a lost packet. Since we know that packet can get lost for two reasons: either because they are damaged in transit or the network is congested and somewhere on the part was insufficient buffer capacity. [8] For this reason, the TCP will enter into congestion avoidance state to try to reduce traffic.

Furthermore, to implement congestion control technique here, we must note that this algorithm must encompass a strategy that must be able to signal the transport that congestion is occurring and the endpoints must also have a policy that decreases the use of network once this signal is received and can also increase the network once the network become stable. According to [3] such algorithm is called multiplicative decrease congestion avoidance:

How it works:

Upon loss of a segment, reduce the congestion window by half down to a minimum of at least one segment. For those segments that remain in the allowed window, back off the retransmission timer exponentially.

With this algorithm, we see that TCP reduces the congestion window by half for every loss.
This decreases the window exponentially thereby reducing the traffic in the network. If the traffic persists, the decrease can even reduce transmission to a single datagram and continue to double timeout values before retransmitting. The main idea is to reduce significantly the traffic reduction to allow routers enough time to clear the datagrams already in their queues.

The Combined Slow Start with Congestion Avoidance Algorithm / source code

Although this two algorithm have different objectives but they are usually combined together to control congestion, In this sense, the sender maintains two state variables for congestion control: a slow start/ congestion window, cwnd, and a threshold size, ssthresh, to switch between the two algorithm. The sender normally sends the minimum of the congestion window and window advertised by the receiver. Once there is a timeout, half of the current cwnd size is stored in ssthresh this is the multiplicative decrease part of the congestion avoidance mentioned above, then cwnd is set to 1 packet (this initiates slow start). Once the sender acknowledged the receipt of new transmission the sender does [8]

```
If (cwnd < ssthresh)
   /* if we're still doing slow start                    [8]
    * open window exponentially */
      Cwnd +=1;
Else
   /* otherwise do Congestion
   *Avoidance increment – by -1 */
   Cwnd +=1 /cwnd;
```

The slow start will then open the window to solve the problem by halving the window that cause the problem, then the congestion avoidance will now follow suit by slowly increasing the window size to probe for more bandwidth becoming available on the path.

VI. TAHOE AND RENO TCP OVERVIEW

Tahoe which was the initial release of TCP that used retransmission scheme mentioned above always waits for the timer to expire before retransmitting. In 1990, TCP Reno was introduced that had changes mainly introducing the new concept of fast recovery and fast Retransmit that has higher throughput especially where only occasional loss occurs. TCP Reno improves retransmission during the fast recovery phase of TCP Reno.

Differences of Tahoe and Reno

Tahoe and Reno have different ways of detecting and handling of lost packets.
Tahoe detects that the packet is lost when the timeout set with the sent packet expires. This means that an Ack is received only after the timeout set had expired. Tahoe will react to this by reducing congestion window to 1 MSS, and reset to slow-start state.
Whereas for Reno, if three duplicate Acks are consecutively received, it indicates there is congestion. Reno will halve the congestion window and quickly perform a "fast retransmit". Reno does not wait for a timeout to expire. Eventually when a timeout is received, Reno enters a phase called fast recovery and slow-start is used as it is with Tahoe.

Reno utilizes the network bandwidth in an efficient way without having to wait for any acknowledgment. Below we will explore more on Reno fast retransmit below.

VII. FAST RETRANSMIT

As mentioned above, fast recovery and fast retransmit are based on the Reno version of TCP. Before describing its functions, we must remember that TCP may generate an immediate duplicate Acknowledgment when segment are received in out of order manner. The purpose of this duplication is to let the sender know that segments sent are received out of order and to inform the sender what sequence

number is expected in the next transmission. We know that TCP does not know whether the duplicate Ack received is caused by a lost of segment or due to reordering of segment. TCP waits until small number duplicate Ack is received. It is assumed that when the problem is about the reordering of segments only two ACKs are sent from the receiver, before the reordered segment are processed and generate a new Ack. Moreover, when three or more duplicate are received is a clear indication that the segment is lost. In this case, TCP will then perform a retransmission of the missing segment without waiting for the timer expiration. We will show the algorithm on how fast retransmit is implemented with fast recovery below.

VIII. FAST RECOVERY

In Reno, congestion avoidance is performed after fast retransmit sends the missing segment, since the lost packet is an indication of possible congestion. This algorithm is called fast recovery. This algorithm has worked remarkably well and believed to have prevented lot congestion on the Internet. In the implementation of this algorithm, slow start is not performed. The reason for not performing slow start is to avoid reducing the flow between the two endpoints abruptly as there are still indications of communication. Since receiver can only generate an Ack when a segments is received, this confirms that the segment sent arrive at the receiver buffer, so slow start will not be necessary in this case.
Fast recovery is believed to be an improvement which has allowed high throughput under reasonable congestions and scaled six orders of magnitude in size, speed, load and connectivity. It has also been relatively efficient at large windows.

The Source Code of Fast Retransmit with Fast Recovery

```
If Number of DUP-ACK received  = 3
        ssthresh =   CWND / 2;
        Retransmit the lost segment ;
        CWND = sshthresh + 3 ;       /*  3 is for 3 DUP-ACKs */

          For each next DUP-ACK
           Increment CWND  by 1.   /* i.e. CWND = CWND + 1.
This inflates the CWND in order to reflect segment that has
left the network */

   If   allowed by CWND and Receiver's Advertised Window
                  Transmit a segment (if any....)

   If Not DUP-ACK      /* i.e. New/Fresh ACK */
            CWND = ssthresh         /* Deflating the window */

          Call Congestion Avoidance Algorithm
```

IX. CONCLUSION

In this seminar paper, we have identified the possible causes of congestion over the network. We have also discussed the four main intertwined algorithms that help to control congestion over the network. We also saw how TCP implements flow controls by having the receiver advertise the amount of data it is willing to accept. The current TCP protocol specifies exponential back off for retransmission timers and congestion avoidance algorithm like slow start. However, we noted the limitation of slow start and its inefficient utilization of network. Finally, we discussed TCP Reno fast retransmit and fast Recovery. We saw that the introduction of TCP Reno changed the way datagram are exchanged. TCP Reno has performed remarkably well and has prevented severe congestion in the Internet. Although these algorithms have great potency in handling congestion, their limitation abounds. More reliable algorithms like RED and others have been designed to handle TCP synchronization, improve throughput and fairness.

X. APPENDIX A

Combine slow start and Congestion Avoidance Algorithm

- ***Initialization for a given connection***
- ***Sets cwnd to one segment and ssthresh to 65535 bytes***
- ***The TCP output routine never sends more than minimum of cwnd and the receiver's advertised window***
- ***When congestion occurs (indicated by a timeout or the reception of duplicate ACKs),***
- ***One-half of the current window size (the minimum of cwnd and the receiver's advertised window, but at least two segments) is saved in ssthresh.***

- ***Additionally, if the congestion is indicated by a timeout, cwnd is set to one segment (slow start)***
- ***When new data is acknowledged by the other end, increase cwnd, but the way it increases depends on whether TCP is performing slow start or congestion avoidance. [3]***

XI. APPENDIX B

The fast retransmit and fast recovery algorithms are usually implemented together as follows

1. When the third duplicate ACK in a row is received,

➔ set ssthresh to one-half the current congestion window, cwnd,

- but no less than two segments.

➔ Retransmit the missing segment.

- Set cwnd to ssthresh plus 3 times the segment size.

This inflates the congestion window by the number of segments that have left the network and which the other end has cached [3]

2. Each time another duplicate ACK arrives,

➔ Increment cwnd by the segment size.

This inflates the congestion window for the additional segment that has left the network. Transmit a packet, if allowed by the new value of cwnd.

3. When the next ACK arrives that acknowledges new data,

➔ Set cwnd to ssthresh (the value set in step 1).

➔ This ACK should be the acknowledgment of the retransmission from step 1, one round-trip time after the retransmission.

➔ Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK.

➔ This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost. Algorithm Adapted from the RFC [3]

REFERENCES

[1] The Great Idea Finder "Fascinating facts about the invention of the Traffic Light William L. Potts "http://www.ideafinder.com/history/inventions/trafficlight.htm Date Accessed: 10th February 2008

[2] D. E. Commer, "Internetworking with TCP/IP, Volume 1: Principles, protocols, and Architecture, 2006

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

[4] http://www.tutorialsweb.com/networking/tcp-ip/images/Fig10_SlidingWindow.jpg

[5] V. Jacobson, "Modified TCP congestion Avoidance Algorithm, "end2end –internet April 30, 1990 ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail

[6] D. X. Wei C. Jin, S. H. Low, S. Hedge, "FAST TCP" Motivation, Architecture, Algorithm, Performance IEEE Network, 2005

[7] V. Jacobson, "Modified TCP congestion Avoidance Algorithm, "end2end –internet April 30, 1990 ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail

[8] http://www.cs.rice.edu/~amsaha/Papers/Cexam/notes/node109.html 15/02/08

[9]http://isi.edu/nsnam/directed_research/dr_wanida/dr/javisin/actionslowstartframe.html Date Accessed 15/02/08