TCP Congestion Control

Abstract

This paper is an exploratory survey of TCP congestion control principles and techniques. In addition to the standard algorithms used in common software implementations of TCP, this paper also describes some of the more common proposals developed by researchers over the years. By studying congestion control techniques used in TCP implementation software and network hardware we can better comprehend the performance issues of packet switched networks and in particular, the public Internet.

1 Introduction

There has been some serious discussion given to the potential of a large-scale Internet collapse due to network overload or congestion [6], [17]. So far the Internet has survived, but there has been a number of incidents throughout the years where serious problems have disabled large parts of the network. Some of these incidents have been a result of algorithms used or not used in the Transmission Control Protocol (TCP) [19]. Others are a result of problems in areas such as security, or perhaps more accurately, the lack thereof [24].

The popularity of the Internet has heightened the need for more bandwidth throughout all tiers of the network. Home users need more bandwidth than the traditional 64Kb/s channel a telephone provider typically allows. Video, music, games, file sharing and browsing the web requires more and more bandwidth to avoid the "World Wide Wait" as it has come to be known by those with slower and often heavily congested connections.

Internet Service Providers (ISPs) who provide the access to the average home customer have had to keep up as more and more users get connected to the information superhighway.

Core backbone providers have had to ramp up their infrastructure to support the increasing demand from their customers below.

Today it would be unusual to find someone in the U.S. that has not heard of the Internet, let alone experienced it in one form or another. The Internet has become the fastest growing technology of all time [8]. So far, the Internet is still chugging along, but a good question to ask is "Will it continue to do so?" Although this paper does not attempt to answer that question, it can help us to understand why it will or why it might not.

Good and bad network performance is largely dependent on the effective implementation of network protocols. TCP, easily the most widely used protocol in the transport layer on

the Internet (e.g. HTTP, TELNET, and SMTP), plays an integral role in determining overall network performance.

Amazingly, TCP has changed very little since its initial design in the early 1980's. A few "tweaks" and "knobs" have been added, but for the most part, the protocol has withstood the test of time. However, there are still a number of performance problems on the Internet and fine tuning TCP software continues to be an area of work for a number of people [21].

Over the past few years, researchers have spent a great deal of effort exploring alternative and additional mechanisms for TCP and related technologies in lieu of potential network overload problems. Some techniques have been implemented; others left behind and still others remain on the drawing board. We'll begin our examination of TCP by trying to understand the underlying design concepts that have made it so successful.

This paper does not cover the basics of the TCP protocol itself, but rather the underlying designs and techniques as they apply to problems of network overload and congestion. For a brief description on the basics of TCP, a companion paper is provided in [14].

2 The End-to-End Argument

The design of TCP was heavily influenced by what has come to be known as the *end-to-end argument* [18]. The key component of the *end-to-end argument* for our purposes is in its method of handling congestion and network overload. The premise of the argument and fundamental to TCP's design is that the end stations are responsible for controlling the rate of data flow. In this model, there are no explicit signaling mechanisms in the network which tell the end stations how fast to transmit, when to transmit, when to speed up or when to slow down. The TCP software in each of the end stations is responsible for answering these questions from implicit knowledge it obtains from the network or the explicit knowledge it receives from the other TCP host.

2.1 An Overview of TCP Flow Control

One of TCP's primary functions is to properly match the transmission rate of the sender to that of the receiver and the network. It is important for the transmission to be at a high enough rate to ensure good performance, but also to protect against overwhelming the network or receiving host.

TCP's 16-bit window field is used by the receiver to tell the sender how many bytes of data the receiver is willing to accept. Since the window field is limited to a maximum of 16 bits, this provides for a maximum window size of 65,535 bytes.

The window size advertised by the receiver tells the sender how much data, starting from the current position in the TCP data byte stream can be sent without waiting for further acknowledgements. As data is sent by the sender and then acknowledged by the receiver, the window slides forward to cover more data in the byte stream. This concept is known as a "sliding window" and is depicted in figure 1 below.

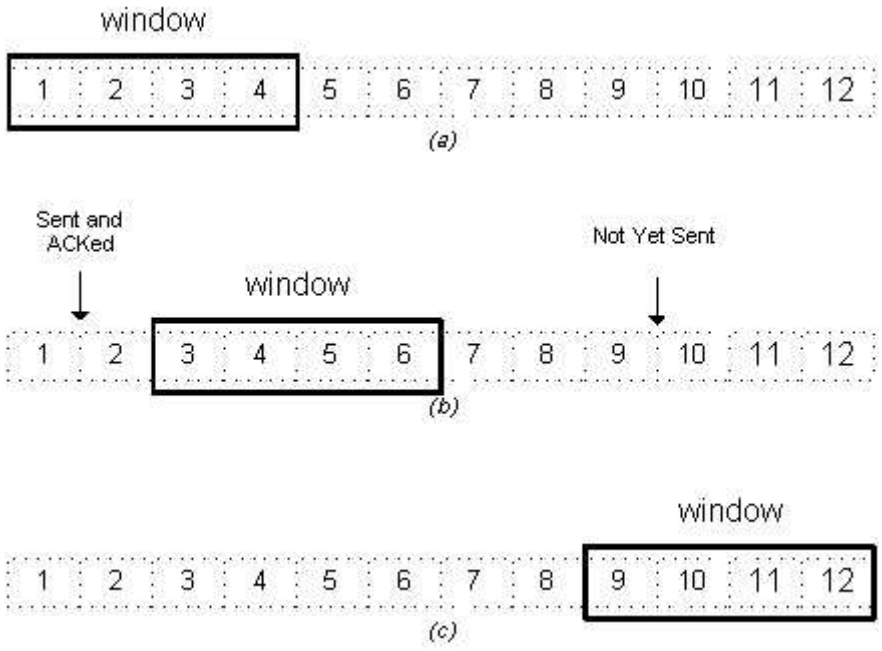


Figure 1 Sliding Window

As shown above, data within the window boundary is eligible to be sent by the sender. Those bytes in the stream prior to the window have already been sent and acknowledged. Bytes ahead of the window have not been sent and must wait for the window to "slide" forward before they can be transmitted by the sender. A receiver can adjust the window size each time it sends acknowledgements to the sender. The maximum transmission rate is ultimately bound by the receiver's ability to accept and process data. However, this technique implies an implicit trust arrangement between the TCP sender and receiver. It has been shown that aggressive or *unfriendly* TCP software implementations can take advantage of this trust relationship to unfairly increase the transmission rate or even to intentionally cause network overload situations [20].

As we will see shortly, the sender and also the network can play a part in determining the transmission rate of data flow as well.

It is important to consider the limitation on the window size of 65,535 bytes. Consider a typical internetwork that may have link speeds of up to 1 Gb/s or more. On a 1 Gb/s network 125,000,000 bytes can be transmitted in one second. This means that if only two

TCP stations are communicating on this link, at best 65,535/125,000,000 or only about .0005 of the bandwidth will be used in each direction each second!

Recognizing the need for larger windows on high-speed networks, the Internet Engineering Task Force released a standard for a "window scale option" defined in RFC 1323 [12]. This standard effectively allows the window to increase from 16 to 32 bits or over 4 billion bytes of data in the window.[1]

2.2 Retransmissions, Timeouts and Duplicate Acknowledgements

TCP is relegated to rely mostly upon implicit signals it learns from the network and remote host. TCP must make an educated guess as to the state of the network and trust the information from the remote host in order to control the rate of data flow. This may seem like an awfully tricky problem, but in most cases TCP handles it in a seemingly simple and straightforward way.

A sender's implicit knowledge of network conditions may be achieved through the use of a *timer*. For each TCP segment sent the sender expects to receive an acknowledgement within some period of time otherwise an error in the form of a timer expiring signals that that something is wrong.

Somewhere in the end-to-end path of a TCP connection a segment can be lost along the way. Often this is due to congestion in network routers where excess packets must be dropped. TCP not only must correct for this situation, but it can also *learn* something about network conditions from it.

Whenever TCP transmits a segment the sender starts a timer which keeps track of how long it takes for an acknowledgment for that segment to return. This timer is known as the *retransmission timer*. If an acknowledgement is returned before the timer expires, which by default is often initialized to 1.5 seconds, the timer is reset with no consequence. If however an acknowledgement for the segment does not return within the timeout period, the sender would retransmit the segment and double the retransmission timer value for each consecutive timeout up to a maximum of about 64 seconds [22]. If there are serious network problems, segments may take a few minutes to be successfully transmitted before the sender eventually times out and generates an error to the sending application.

Fundamental to the timeout and retransmission strategy of TCP is the measurement of the *round-trip time* between two communicating TCP hosts. The round-trip time may vary during the TCP connection as network traffic patterns fluctuate and as routes become available or unavailable.

---

[1] A TCP option negotiated in the TCP connection establishment phase sets the number of bits by which the window is right-shifted in order to increase the value of the window.

TCP keeps track of when data is sent and at what time acknowledgements covering those sent bytes are returned. TCP uses this information to calculate an estimate of round trip time. As packets are sent and acknowledged, TCP adjusts its round-trip time estimate and uses this information to come up with a reasonable timeout value for packets sent. If acknowledgements return quickly, the round-trip time is short and the retransmission timer is thus set to a lower value. This allows TCP to quickly retransmit data when network response time is good, alleviating the need for a long delay between the occasional lost segment. The converse is also true. TCP does not retransmit data too quickly during times when network response time is long.

If a TCP data segment is lost in the network, a receiver will never even know it was once sent. However, the sender is waiting for an acknowledgement for that segment to return. In one case, if an acknowledgement doesn't return, the sender's retransmission timer expires which causes a retransmission of the segment. If however the sender had sent at least one additional segment after the one that was lost and that later segment is received correctly, the receiver does not send an acknowledgement for the later, out of order segment.

The receiver cannot acknowledgement out of order data; it must acknowledge the last contiguous byte it has received in the byte stream prior to the lost segment. In this case, the receiver will send an acknowledgement indicating the last contiguous byte it has received. If that last contiguous byte was already acknowledged, we call this a duplicate ACK. The reception of duplicate ACKs can implicitly tell the sender that a segment may have been lost or delayed. The sender knows this because the receiver only generates a duplicate ACK when it receives other, out of order segments. In fact, the Fast Retransmit algorithm described later uses duplicate ACKs as a way of speeding up the retransmission process.

3.0 Standard TCP Congestion Control Algorithms

The standard fare in TCP implementations today can be found in RFC 2581 [2]. This reference document specifies four standard congestion control algorithms that are now in common use. Each of the algorithms noted within that document was actually designed long before the standard was published [9], [11]. Their usefulness has passed the test of time.

The four algorithms, Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery are described below.

3.1 Slow Start

Slow Start, a requirement for TCP software implementations is a mechanism used by the sender to control the transmission rate, otherwise known as sender-based flow control. This is accomplished through the return rate of acknowledgements from the receiver. In

other words, the rate of acknowledgements returned by the receiver determine the rate at which the sender can transmit data.

When a TCP connection first begins, the Slow Start algorithm initializes a *congestion window* to one segment, which is the maximum segment size (MSS) initialized by the receiver during the connection establishment phase. When acknowledgements are returned by the receiver, the congestion window increases by one segment for each acknowledgement returned. Thus, the sender can transmit the minimum of the congestion window and the advertised window of the receiver, which is simply called the *transmission window*.

Slow Start is actually not very slow when the network is not congested and network response time is good. For example, the first successful transmission and acknowledgement of a TCP segment increases the window to two segments. After successful transmission of these two segments and acknowledgements completes, the window is increased to four segments. Then eight segments, then sixteen segments and so on, doubling from there on out up to the maximum window size advertised by the receiver or until congestion finally does occur.

At some point the congestion window may become too large for the network or network conditions may change such that packets may be dropped. Packets lost will trigger a timeout at the sender. When this happens, the sender goes into congestion avoidance mode as described in the next section.

3.2 Congestion Avoidance

During the initial data transfer phase of a TCP connection the Slow Start algorithm is used. However, there may be a point during Slow Start that the network is forced to drop one or more packets due to overload or congestion. If this happens, Congestion Avoidance is used to slow the transmission rate. However, Slow Start is used in conjunction with Congestion Avoidance as the means to get the data transfer going again so it doesn't slow down and stay slow.

In the Congestion Avoidance algorithm a retransmission timer expiring or the reception of duplicate ACKs can implicitly signal the sender that a network congestion situation is occurring. The sender immediately sets its transmission window to one half of the current window size (the minimum of the congestion window and the receiver's advertised window size), but to at least two segments. If congestion was indicated by a timeout, the congestion window is reset to one segment, which automatically puts the sender into Slow Start mode. If congestion was indicated by duplicate ACKs, the Fast Retransmit and Fast Recovery algorithms are invoked (see below).

As data is received during Congestion Avoidance, the congestion window is increased. However, Slow Start is only used up to the halfway point where congestion originally

occurred. This halfway point was recorded earlier as the new transmission window. After this halfway point, the congestion window is increased by one segment for all segments in the transmission window that are acknowledged. This mechanism will force the sender to more slowly grow its transmission rate, as it will approach the point where congestion had previously been detected.

3.3 Fast Retransmit

When a duplicate ACK is received, the sender does not know if it is because a TCP segment was lost or simply that a segment was delayed and received out of order at the receiver. If the receiver can re-order segments, it should not be long before the receiver sends the latest expected acknowledgement. Typically no more than one or two duplicate ACKs should be received when simple out of order conditions exist. If however more than two duplicate ACKs are received by the sender, it is a strong indication that at least one segment has been lost. The TCP sender will assume enough time has lapsed for all segments to be properly re-ordered by the fact that the receiver had enough time to send three duplicate ACKs.

When three or more duplicate ACKs are received, the sender does not even wait for a retransmission timer to expire before retransmitting the segment (as indicated by the position of the duplicate ACK in the byte stream). This process is called the Fast Retransmit algorithm and was first defined in [11]. Immediately following Fast Retransmit is the Fast Recovery algorithm.

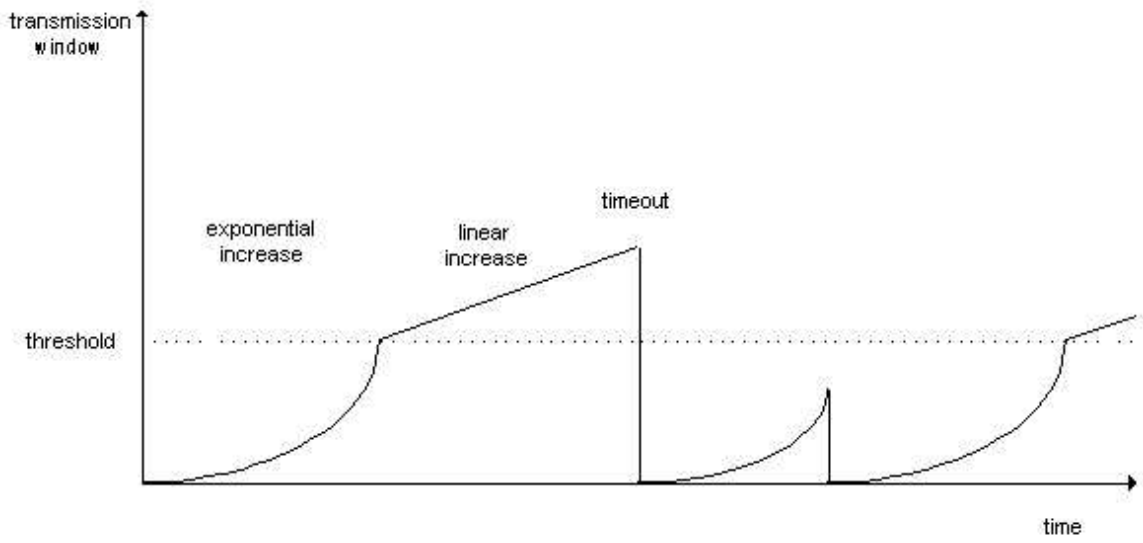
3.4 Fast Recovery

Since the Fast Retransmit algorithm is used when duplicate ACKs are being received, the TCP sender has implicit knowledge that there is data still flowing to the receiver. Why? The reason is because duplicate ACKs can only be generated when a segment is received. This is a strong indication that serious network congestion may not exist and that the lost segment was a rare event. So instead of reducing the flow of data abruptly by going all the way into Slow Start, the sender only enters Congestion Avoidance mode.

Rather than start at a window of one segment as in Slow Start mode, the sender resumes transmission with a larger window, incrementing as if in Congestion Avoidance mode. This allows for higher throughput under the condition of only moderate congestion [23].

To summarize this section of the paper, figure 2 below depicts what a typical TCP data transfer phase using TCP congestion control might look like. Notice the periods of exponential window size increase, linear increase and drop-off. Each of these scenarios depicts the sender's response to implicit or explicit signals it receives about network conditions.

Figure 2 Congestion Control Overview



4.0 Latest Techniques

Although RFC 2581 and its associated algorithms have been doing an excellent job in ensuring top performance in lieu of congestion on TCP/IP networks, there are still a lot of work going into enhancing TCP performance and responsiveness to congestion. During the 1990's researchers such as Sally Floyd, Van Jacobson, Mark Allman, W. Richard Stevens, Jamshid Mahdavi and a host of others starting producing a massive amount of research and experiments with TCP and related congestion control ideas. The wealth of information in this area is really phenomenal and it is hard to pick out some of the best ideas to present in this paper. Nevertheless, this section is an attempt to provide an overview of some of those popular ideas over the last decade. TCP and congestion control on the Internet is an area that is still actively being researched. For more information, consult the references noted in this paper.

4.1 Selective Acknowledgements

Whenever a TCP segment has been sent and the sender's retransmission timer expires, the sender is forced to retransmit the segment, which the sender assumes has been lost. However, it is possible that between the time when the segment was initially sent and the time when the retransmission window expired, other segments in the window may have been sent after the lost segment. It is also possible that these later segments arrived at the receiver and are simply queued awaiting the missing segment so they can be properly re-ordered. The receiver has no way of informing the sender that it has received other segments because of the requirement to acknowledgement only the contiguous bytes it has received. This case demonstrates a potential inefficiency in the way TCP handles the occasional loss of segments.

Ideally, the sender should only retransmit the lost segment(s) while the receiver continues to queue the later segments. This behavior was identified as a potential improvement in TCP's congestion control algorithms as early as 1988 [10]. It was only until recently that a mechanism to retransmit just the lost segments in these situations was put into standard TCP implementations [15], [16].

Selective Acknowledgement (or SACK) is this technique implemented as a TCP option that can help reduce unnecessary retransmissions on the part of the sender. If the TCP connection has negotiated the use of SACK (through the use of the TCP header option fields), the receiver can offer feedback to the sender in the form of the selective acknowledgement option. The receiver reports to the sender, which *blocks* of data have arrived using the format show in figure 3 below.

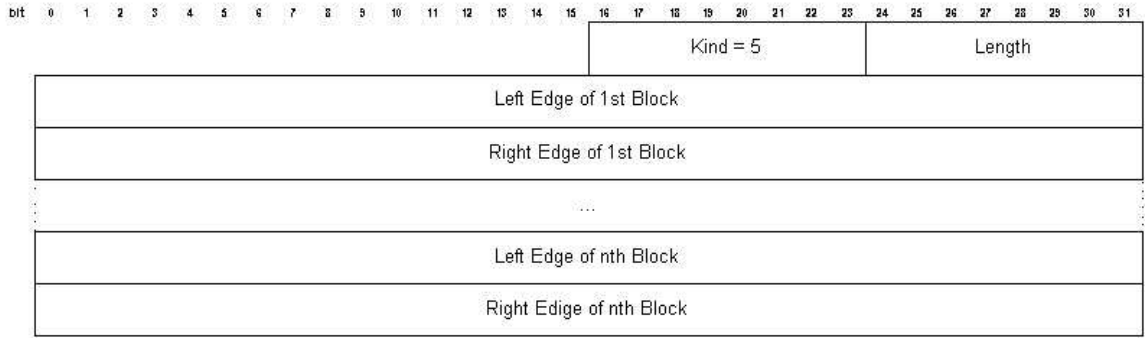


Figure 3 SACK Option

This list of blocks in the SACK option tells the sender which contiguous byte stream blocks it has received. At maximum, four SACK blocks can be sent in one TCP segment because of the maximum size of the options field in a TCP head is 40 bytes and each block report consists of 8 bytes plus the option header field of 4 bytes (for a total of 36 bytes).

Note that the SACK information is advisory information only. The sender cannot rely upon the receiver to maintain the out-of-order data. Obviously the performance gain is to be had when the receiver does queue and re-order data that has been reported with the SACK option so that the sender limits its retransmissions.

4.2 NewReno

The Tahoe implementation of BSD included the ability to do Slow Start, Congestion Avoidance and Fast Recovery. Reno was the implementation of TCP that included the Tahoe implementation plus the ability to do Fast Retransmit. NewReno is a slight modification to the Reno implementation of TCP [7] that can improve performance during Fast Recovery and Fast Retransmit mode.[2]

[2] The original names of Reno and Tahoe were derived from the names of BSD TCP/IP implementations.

The NewReno implementation only applies if SACK has not been negotiated in a connection. From [7], an overview of the NewReno is as follows:

> *In the absence of SACK, there is little information available to the TCP sender in making retransmission decisions during Fast Recovery. From the three duplicate acknowledgements, the sender infers a packet loss, and retransmits the indicated packet. After this, the data sender could receive additional duplicate acknowledgements, as the data receiver acknowledges additional data packets that were already in flight when the sender entered Fast Retransmit.*
>
> *In the case of multiple packets dropped from a single window of data, the first new information available to the sender comes when the sender receives an acknowledgement for the retransmitted packet (that is the packet retransmitted when Fast Retransmit was first entered). If there had been a single packet drop, then the acknowledgement for this packet will acknowledge all of the packets transmitted before Fast Retransmit was entered (in the absence of reordering). However, when there were multiple packet drops, then the acknowledgement for the retransmitted packet will acknowledge some but not all of the packets transmitted before the Fast Retransmit. We call this packet a partial acknowledgment.*

According to RFC 2582 above, the TCP sender should infer from the partial acknowledgement that the indicated segment has been lost and immediately retransmit the segment.[3]

4.3 Other TCP Congestion Control Techniques

There are a number of other proposals and experiments that have been performed on TCP to improve performance. In this section, we will just briefly cover two of the most recent from some of the leading researchers in the field.

4.3.1 Increasing TCP's Initial Window Size

The experimental RFC 2414 [Allman98] suggests increasing TCP's initial window size from one segment to roughly 4 kilobytes. By doing so, in certain situations it is believed to offer better performance by being able to fill the "pipe" quicker.

4.3.2 TCP Pacing

If a TCP sender, a router or other intermediate device space TCP packets apart, the bursty nature of data transmission may be reduced. The intended affect is that by reducing the

[3] We use the term *segment* in place of *packet* as is used in RFC 2582, but essentially the meanings of the two are the same for our purposes here.

bursts in network traffic, periods of congestion and eventual packet loss are also reduced. Its advantages and disadvantages are only beginning to be understood [1]. However, a popular commercial product already implements a technique similar to TCP pacing and is being installed in many large organizations [Packeteer00].

4.4 Non-TCP Congestion Control Techniques

There are number of techniques which are worth our time to examine even though they are not directly implemented within TCP software on end systems. These techniques can indirectly affect TCP congestion control.

For example, whenever a router drops a packet, it in effect is providing a signal to the TCP sender to slow down by causing a retransmission timer to expire. If routers could use some advanced packet drop techniques, they may be able to better control network congestion through the implicit signals TCP senders detect.

Also, there are non-technical designs that can affect network performance. By implementing a system where a cost is associated with network transmission, end stations may adjust their transmission rate up or down based on the value of performance they may require. Both of these types of techniques are briefly explored below.

4.4.1 Random Early Detection

Perhaps one of the most notable enhancements to congestion control techniques has been the development of Random Early Detection (RED) for internetwork routers. This algorithm manages router queues and drops packets based on a queue threshold. This in effect causes congestion control to be activated just prior to any actual network congestion event.

For example, to signal traditionally implemented TCP senders to slow down, a router using RED will monitor its average queue depth. As network traffic increases and crosses a threshold, RED will begin to drop packets with a certain probability. TCP senders will go into Slow Start and Congestion Avoidance mode once they have detected a lost packet. This helps the network slow down before actual congestion occurs.

The beauty of this technique is that is fairly simple to implement and it helps prevent high bandwidth TCP connections from starving low bandwidth TCP connections. It also does not allow unfriendly TCP implementations to gain an unfair advantage by removing the sole reliance on the TCP sender/receiver trust relationship.[4] Since the packet drop function is based on a certain probability, connections using a larger share of the bandwidth will have more of their traffic dropped than low bandwidth users.

[4] Such as in the case where TCP senders must rely on explicit window and ACK information from the TCP receivers.

RED was first described in [4] and is recommended in [3].

4.4.2 Explicit Congestion Notification

Also briefly described in [4] and further expanded upon in [5], Explicit Congestion Notification (ECN) is a technique that just marks packets instead of dropping them as RED usually does. The idea behind implementing ECN instead of RED is to avoid packet drops, particularly where the delay involved caused by retransmission needs to be avoided. Good examples of cases where this delay should be avoided are with real-time applications such as two-way voice communications or when using a terminal program such as TELNET.

Routers can mark two bits in the IP Type of Service (ToS) header field to signal whether or not congestion is occurring. TCP senders can then adjust their rate of transmission appropriately if they see that these bits are set to indicate a network congestion condition is occurring.

4.4.3 Network Pricing

An entirely different category of congestion control is through the use of a network-pricing model. In this case, the cost of transmission either in time, usage or capacity can be on a fee basis. By making the transmission of TCP non-free, there may be a monetary incentive to avoid congestion [13]. Applying a cost on transmission may help force senders to minimize the amount of traffic they generate. This in effect attempts to make it expensive for users to cause congestion and high load conditions. It is analogous to getting a speeding ticket on the highway.

5.0 Conclusion

Over the past decade a large amount of research and experimentation has gone into TCP performance and congestion control. A great deal of that work has paid off in the form of an Internet that continues to function considerably well even in light of the increasing traffic demands.

Now of great concern to a number of network practitioners is the concept of "network fairness". Here the goal is to provide some level playing field for all participants, and to avoid the greedy or "eager" TCP senders to make room for low bandwidth connections. The use of RED is one mechanism that is becoming popular among Internet Service Providers and large organizations.

It remains to be seen how far current congestion control techniques can carry the Internet as its growth continues. So far they have performed admirably.

Abbreviations

ACK	Acknowledgement
bit	binary digit
BSD	Berkely Software Distribution
ECN	Explicit Congestion Notification
Gb/s	Gibabits per second
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Provider
Kb/s	Kilobits per second
MSS	Maximum Segment Size
RED	Random Early Detection
RFC	Request For Comments
RSVP	Resource ReSerVation Protocol
SACK	Selective ACKnowledgement
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
ToS	Type of Service
UDP	User Datagram Protocol

References

[1] Amit Aggarwal, Stefan Savage, and Thomas Anderson. Understanding the Performance of TCP Pacing, March 30, 2000, IEEE InfoCom 2000.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control, April 1999, RFC 2581.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Patridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. WrocLawski, and Lixia Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet, April 1998, RFC 2309.

[4] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking, August 1993.

[5] Sally Floyd. TCP and Explicit Congestion Notification, ACM Computer Communications Review, October 1994, p. 10-23.

[6] Sally Floyd and Kevin Fall. Promoting the Use of End-to-End Congestion Control in Internet. IEEE/ACM Transactions on Networking, August 1999.

[7] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Mechanism, April 1999, RFC 2582.

[8] Harris Interactive. P.C. and Internet Use Continue to Grow at Record Pace. Press Release, February 7, 2000.

[9] Van Jacobson. Congestion Avoidance and Control. Computer Communications Review, Volume 18 number 4, pp. 314-329, August 1988.

[10] V. Jacobson and R. Braden. TCP Extensions for Long-Delay Paths, October 1988, RFC 1072.

[11] Van Jacobson. Modified TCP Congestion Control Avoidance Algorithm. end-2-end-interest mailing list, April 30, 1990.

[12] V. Jacobson, R. Braden and D. Borman. TCP Extensions for High Performance, May 1992, RFC 1323.

[13] Scott Jordan. Pricing and Differentiated Services in Internet and ATM, http://www.eng.uci.edu/~sjordan/pubs/Pricing/index.htm, March 11, 1999.

[14] John Kristoff. The Transmission Control Protocol, March 2000.

[15] Jamshid Mahdavi. Private e-mail to John Kristoff, December 12, 1999.

[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options, October 1996, RFC 2018.

[17] Bob Metcalfe. From the Ether. InfoWorld, December 4, 1995.

[18] David D. Clark. The Design Philosophy of the DARPA Internet Protocols. In Proceedings SIGCOMM ' 88, Computer Communications Review Vol. 18, No. 4, August 1988, pp. 106-114).

[19] Jon Postel. Transmission Control Protocol, September 1981, RFC 793.

[20] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson, TCP Congestion Control with a Misbehaving Receiver, ACM Computer Communications Review, October 1999.

[21] Jeffrey Semke, Jamshid Mahdavi and Matthew Mathis. Automatic TCP Buffer Tuning, Computer Communications Review, ACM SIGCOMM, Volume 28, Number 4, October 1998.

[22] W. Richard Stevens. TCP/IP Illustrated, Volume 1: The Protocols. Addison Wesley, ISBN: 0-201-63346-9. January 1994.

[23] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997, RFC 2001.

[24] Bob Sullivan. Remembering the Net Crash of '88, MSNBC, November 2 1998.