Network Working Group
Request for Comments: 1106

R. Fox
Tandem

June 1989

TCP Big Window and Nak Options

Status of this Memo

This memo discusses two extensions to the TCP protocol to provide a more efficient operation over a network with a high bandwidth*delay product. The extensions described in this document have been implemented and shown to work using resources at NASA. This memo describes an Experimental Protocol, these extensions are not proposed as an Internet standard, but as a starting point for further research. Distribution of this memo is unlimited.

Abstract

Two extensions to the TCP protocol are described in this RFC in order to provide a more efficient operation over a network with a high bandwidth*delay product. The main issue that still needs to be solved is congestion versus noise. This issue is touched on in this memo, but further research is still needed on the applicability of the extensions in the Internet as a whole infrastructure and not just high bandwidth*delay product networks. Even with this outstanding issue, this document does describe the use of these options in the isolated satellite network environment to help facilitate more efficient use of this special medium to help off load bulk data transfers from links needed for interactive use.

1. Introduction

Recent work on TCP has shown great performance gains over a variety of network paths [1]. However, these changes still do not work well over network paths that have a large round trip delay (satellite with a 600 ms round trip delay) or a very large bandwidth (transcontinental DS3 line). These two networks exhibit a higher bandwidth*delay product, over 10**6 bits, than the 10**5 bits that TCP is currently limited to. This high bandwidth*delay product refers to the amount of data that may be unacknowledged so that all of the networks bandwidth is being utilized by TCP. This may also be referred to as "filling the pipe" [2] so that the sender of data can always put data onto the network and the receiver will always have something to read, and neither end of the connection will be forced to wait for the other end.

After the last batch of algorithm improvements to TCP, performance

over high bandwidth*delay networks is still very poor. It appears that no algorithm changes alone will make any significant improvements over high bandwidth*delay networks, but will require an extension to the protocol itself. This RFC discusses two possible options to TCP for this purpose.

The two options implemented and discussed in this RFC are:

1. NAKs

   This extension allows the receiver of data to inform the sender that a packet of data was not received and needs to be resent. This option proves to be useful over any network path (both high and low bandwidth*delay type networks) that experiences periodic errors such as lost packets, noisy links, or dropped packets due to congestion. The information conveyed by this option is advisory and if ignored, does not have any effect on TCP what so ever.

2. Big Windows

   This option will give a method of expanding the current 16 bit (64 Kbytes) TCP window to 32 bits of which 30 bits (over 1 gigabytes) are allowed for the receive window. (The maximum window size allowed in TCP due to the requirement of TCP to detect old data versus new data. For a good explanation please see [2].) No changes are required to the standard TCP header [6]. The 16 bit field in the TCP header that is used to convey the receive window will remain unchanged. The 32 bit receive window is achieved through the use of an option that contains the upper half of the window. It is this option that is necessary to fill large data pipes such as a satellite link.

This RFC is broken up into the following sections: section 2 will discuss the operation of the NAK option in greater detail, section 3 will discuss the big window option in greater detail. Section 4 will discuss other effects of the big windows and nak feature when used together. Included in this section will be a brief discussion on the effects of congestion versus noise to TCP and possible options for satellite networks. Section 5 will be a conclusion with some hints as to what future development may be done at NASA, and then an appendix containing some test results is included.

2. NAK Option

Any packet loss in a high bandwidth*delay network will have a catastrophic effect on throughput because of the simple acknowledgement of TCP. TCP always acks the stream of data that has

successfully been received and tells the sender the next byte of data of the stream that is expected. If a packet is lost and succeeding packets arrive the current protocol has no way of telling the sender that it missed one packet but received following packets. TCP currently resends all of the data over again, after a timeout or the sender suspects a lost packet due to a duplicate ack algorithm [1], until the receiver receives the lost packet and can then ack the lost packet as well as succeeding packets received. On a normal low bandwidth*delay network this effect is minimal if the timeout period is set short enough. However, on a long delay network such as a T1 satellite channel this is catastrophic because by the time the lost packet can be sent and the ack returned the TCP window would have been exhausted and both the sender and receiver would be temporarily stalled waiting for the packet and ack to fully travel the data pipe. This causes the pipe to become empty and requires the sender to refill the pipe after the ack is received. This will cause a minimum of 3*X bandwidth loss, where X is the one way delay of the medium and may be much higher depending on the size of the timeout period and bandwidth*delay product. Its 1X for the packet to be resent, 1X for the ack to be received and 1X for the next packet being sent to reach the destination. This calculation assumes that the window size is much smaller than the pipe size (window = 1/2 data pipe or 1X), which is the typical case with the current TCP window limitation over long delay networks such as a T1 satellite link.

An attempt to reduce this wasted bandwidth from 3*X was introduced in [1] by having the sender resend a packet after it notices that a number of consecutively received acks completely acknowledges already acknowledged data. On a typical network this will reduce the lost bandwidth to almost nil, since the packet will be resent before the TCP window is exhausted and with the data pipe being much smaller than the TCP window, the data pipe will not become empty and no bandwidth will be lost. On a high delay network the reduction of lost bandwidth is minimal such that lost bandwidth is still significant. On a very noisy satellite, for instance, the lost bandwidth is very high (see appendix for some performance figures) and performance is very poor.

There are two methods of informing the sender of lost data. Selective acknowledgements and NAKS. Selective acknowledgements have been the object of research in a number of experimental protocols including VMTP [3], NETBLT [4], and SatFTP [5]. The idea behind selective acks is that the receiver tells the sender which pieces it received so that the sender can resend the data not acked but already sent once. NAKs on the other hand, tell the sender that a particular packet of data needs to be resent.

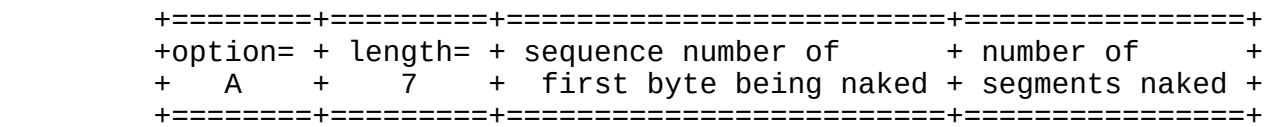
There are a couple of disadvantages of selective acks. Namely, in

some of the protocols mentioned above, the receiver waits a certain time before sending the selective ack so that acks may be bundled up. This delay can cause some wasted bandwidth and requires more complex state information than the simple nak. Even if the receiver doesn't bundle up the selective acks but sends them as it notices that packets have been lost, more complex state information is needed to determine which packets have been acked and which packets need to be resent. With naks, only the immediate data needed to move the left edge of the window is naked, thus almost completely eliminating all state information.

The selective ack has one advantage over naks. If the link is very noisy and packets are being lost close together, then the sender will find out about all of the missing data at once and can send all of the missing data out immediately in an attempt to move the left window edge in the acknowledge number of the TCP header, thus keeping the data pipe flowing. Whereas with naks, the sender will be notified of lost packets one at a time and this will cause the sender to process extra packets compared to selective acks. However, empirical studies has shown that most lost packets occur far enough apart that the advantage of selective acks over naks is rarely seen. Also, if naks are sent out as soon as a packet has been determined lost, then the advantage of selective acks becomes no more than possibly a more aesthetic algorithm for handling lost data, but offers no gains over naks as described in this paper. It is this reason that the simplicity of naks was chosen over selective acks for the current implementation.

2.1 Implementation details

When the receiver of data notices a gap between the expected sequence number and the actual sequence number of the packet received, the receiver can assume that the data between the two sequence numbers is either going to arrive late or is lost forever. Since the receiver can not distinguish between the two events a nak should be sent in the TCP option field. Naking a packet still destined to arrive has the effect of causing the sender to resend the packet, wasting one packets worth of bandwidth. Since this event is fairly rare, the lost bandwidth is insignificant as compared to that of not sending a nak when the packet is not going to arrive. The option will take the form as follows:



This option contains the first sequence number not received and a

count of how many segments of bytes needed to be resent, where segments is the size of the current TCP MSS being used for the connection. Since a nak is an advisory piece of information, the sending of a nak is unreliable and no means for retransmitting a nak is provided at this time.

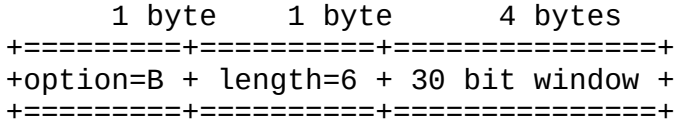
When the sender of data receives the option it may either choose to do nothing or it will resend the missing data immediately and then continue sending data where it left off before receiving the nak. The receiver will keep track of the last nak sent so that it will not repeat the same nak. If it were to repeat the same nak the protocol could get into the mode where on every reception of data the receiver would nak the first missing data frame. Since the data pipe may be very large by the time the first nak is read and responded to by the sender, many naks would have been sent by the receiver. Since the sender does not know that the naks are repetitious it will resend the data each time, thus wasting the network bandwidth with useless retransmissions of the same piece of data. Having an unreliable nak may result in a nak being damaged and not being received by the sender, and in this case, we will let the tcp recover by its normal means. Empirical data has shown that the likelihood of the nak being lost is quite small and thus, this advisory nak option works quite well.

3. Big Window Option

Currently TCP has a 16 bit window limitation built into the protocol. This limits the amount of outstanding unacknowledged data to 64 Kbytes. We have already seen that some networks have a pipe larger than 64 Kbytes. A T1 satellite channel and a cross country DS3 network with a 30ms delay have data pipes much larger than 64 Kbytes. Thus, even on a perfectly conditioned link with no bandwidth wasted due to errors, the data pipe will not be filled and bandwidth will be wasted. What is needed is the ability to send more unacknowledged data. This is achieved by having bigger windows, bigger than the current limitation of 16 bits. This option to expands the window size to 30 bits or over 1 gigabytes by literally expanding the window size mechanism currently used by TCP. The added option contains the upper 15 bits of the window while the lower 16 bits will continue to go where they normally go [6] in the TCP header.

A TCP session will use the big window options only if both sides agree to use them, otherwise the option is not used and the normal 16 bit windows will be used. Once the 2 sides agree to use the big windows then every packet thereafter will be expected to contain the window option with the current upper 15 bits of the window. The negotiation to decide whether or not to use the bigger windows takes place during the SYN and SYN ACK segments of the TCP connection

startup process. The originator of the connection will include in the SYN segment the following option:



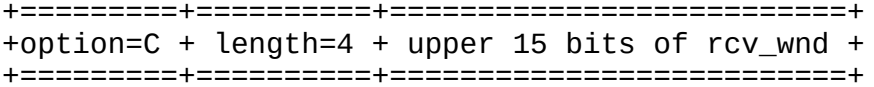
If the other end of the connection wants to use big windows it will include the same option back in the SYN ACK segment that it must send. At this point, both sides have agreed to use big windows and the specified windows will be used. It should be noted that the SYN and SYN ACK segments will use the small windows, and once the big window option has been negotiated then the bigger windows will be used.

Once both sides have agreed to use 32 bit windows the protocol will function just as it did before with no difference in operation, even in the event of lost packets. This claim holds true since the rcv_wnd and snd_wnd variables of tcp contain the 16 bit windows until the big window option is negotiated and then they are replaced with the appropriate 32 bit values. Thus, the use of big windows becomes part of the state information kept by TCP.

Other methods of expanding the windows have been presented, including a window multiple [2] or streaming [5], but this solution is more elegant in the sense that it is a true extension of the window that one day may easily become part of the protocol and not just be an option to the protocol.

3.1 How does it work

Once a connection has decided to use big windows every succeeding packet must contain the following option:



With all segments sent, the sender supplies the size of its receive window. If the connection is only using 16 bits then this option is not supplied, otherwise the lower 16 bits of the receive window go into the tcp header where it currently resides [6] and the upper 15 bits of the window is put into the data portion of the option C. When the receiver processes the packet it must first reform the window and then process the packet as it would in the absence of the option.

3.2 Impact of changes

In implementing the first version of the big window option there was very little change required to the source. State information must be added to the protocol to determine if the big window option is to be used and all 16 bit variables that dealt with window information must now become 32 bit quantities. A future document will describe in more detail the changes required to the 4.3 bsd tcp source code. Test results of the window change only are presented in the appendix. When expanding 16 bit quantities to 32 bit quantities in the TCP control block in the source (4.3 bsd source) may cause the structure to become larger than the mbuf used to hold the structure. Care must be taken to insure this doesn't occur with your system or undetermined events may take place.

4. Effects of Big Windows and Naks when used together

With big windows alone, transfer times over a satellite were quite impressive with the absence of any introduced errors. However, when an error simulator was used to create random errors during transfers, performance went down extremely fast. When the nak option was added to the big window option performance in the face of errors went up some but not to the level that was expected. This section will discuss some issues that were overcome to produce the results given in the appendix.

4.1 Window Size and Nak benefits

With out errors, the window size required to keep the data pipe full is equal to the round trip delay * throughput desired, or the data pipe bandwidth (called Z from now on). This and other calculations assume that processing time of the hosts is negligible. In the event of an error (without NAKs), the window size needs to become larger than Z in order to keep the data pipe full while the sender is waiting for the ack of the resent packet. If the window size is equaled to Z and we assume that the retransmission timer is equaled to Z, then when a packet is lost, the retransmission timer will go off as the last piece of data in the window is sent. In this case, the lost piece of data can be resent with no delay. The data pipe will empty out because it will take 1/2Z worth of data to get the ack back to the sender, an additional 1/2Z worth of data to get the data pipe refilled with new data. This causes the required window to be 2Z, 1Z to keep the data pipe full during normal operations and 1Z to keep the data pipe full while waiting for a lost packet to be resent and acked.

If the same scenario in the last paragraph is used with the addition of NAKs, the required window size still needs to be 2Z to avoid

wasting any bandwidth in the event of a dropped packet. This appears to mean that the nak option does not provide any benefits at all. Testing showed that the retransmission timer was larger than the data pipe and in the event of errors became much bigger than the data pipe, because of the retransmission backoff. Thus, the nak option bounds the required window to 2Z such that in the event of an error there is no lost bandwidth, even with the retransmission timer fluctuations. The results in the appendix shows that by using naks, bandwidth waste associated with the retransmission timer facility is eliminated.

4.2 Congestions vs Noise

An issue that must be looked at when implementing both the NAKs and big window scheme together is in the area of congestion versus lost packets due to the medium, or noise. In the recent algorithm enhancements [1], slow start was introduced so that whenever a data transfer is being started on a connection or right after a dropped packet, the effective send window would be set to a very small size (typically would equal the MSS being used). This is done so that a new connection would not cause congestion by immediately overloading the network, and so that an existing connection would back off the network if a packet was dropped due to congestion and allow the network to clear up. If a connection using big windows loses a packet due to the medium (a packet corrupted by an error) the last thing that should be done is to close the send window so that the connection can only send 1 packet and must use the slow start algorithm to slowly work itself back up to sending full windows worth of data. This algorithm would quickly limit the usefulness of the big window and nak options over lossy links.

On the other hand, if a packet was dropped due to congestion and the sender assumes the packet was dropped because of noise the sender will continue sending large amounts of data. This action will cause the congestion to continue, more packets will be dropped, and that part of the network will collapse. In this instance, the sender would want to back off from sending at the current window limit. Using the current slow start mechanism over a satellite builds up the window too slowly [1]. Possibly a better solution would be for the window to be opened 2*Rlog2(W) instead of R*log2(W) [1] (open window by 2 packets instead of 1 for each acked packet). This will reduce the wasted bandwidth by opening the window much quicker while giving the network a chance to clear up. More experimentation is necessary to find the optimal rate of opening the window, especially when large windows are being used.

The current recommendation for TCP is to use the slow start mechanism in the event of any lost packet. If an application knows that it

will be using a satellite with a high error rate, it doesn't make sense to force it to use the slow start mechanism for every dropped packet. Instead, the application should be able to choose what action should happen in the event of a lost packet. In the BSD environment, a setsockopt call should be provided so that the application may inform TCP to handle lost packets in a special way for this particular connection. If the known error rate of a link is known to be small, then by using slow start with modified rate from above, will cause the amount of bandwidth loss to be very small in respect to the amount of bandwidth actually utilized. In this case, the setsockopt call should not be used. What is really needed is a way for a host to determine if a packet or packets are being dropped due to congestion or noise. Then, the host can choose to do the right thing. This will require a mechanism like source quench to be used. For this to happen more experimentation is necessary to determine a solid definition on the use of this mechanism. Now it is believed by some that using source quench to avoid congestion only adds to the problem, not help suppress it.

The TCP used to gather the results in the appendix for the big window with nak experiment, assumed that lost packets were the result of noise and not congestion. This assumption was used to show how to make the current TCP work in such an environment. The actual satellite used in the experiment (when the satellite simulator was not used) only experienced an error rate around 10e-10. With this error rate it is suggested that in practice when big windows are used over the link, TCP should use the slow start mechanism for all lost packets with the 2*Rlog2(W) rate discussed above. Under most situations when long delay networks are being used (transcontinental DS3 networks using fiber with very low error rates, or satellite links with low error rates) big windows and naks should be used with the assumption that lost packets are the result of congestion until a better algorithm is devised [7].

Another problem noticed, while testing the affects of slow start over a satellite link, was at times, the retransmission timer was set so restrictive, that milliseconds before a naked packet's ack is received the retransmission timer would go off due to a timed packet within the send window. The timer was set at the round trip delay of the network allowing no time for packet processing. If this timer went off due to congestion then backing off is the right thing to do, otherwise to avoid the scenario discovered by experimentation, the transmit timer should be set a little longer so that the retransmission timer does not go off too early. Care must be taken to make sure the right thing is done in the implementation in question so that a packet isn't retransmitted too soon, and blamed on congestion when in fact, the ack is on its way.

4.3 Duplicate Acks

Another problem found with the 4.3bsd implementation is in the area of duplicate acks. When the sender of data receives a certain number of acks (3 in the current Berkeley release) that acknowledge previously acked data before, it then assumes that a packet has been lost and will resend the one packet assumed lost, and close its send window as if the network is congested and the slow start algorithm mention above will be used to open the send window. This facility is no longer needed since the sender can use the reception of a nak as its indicator that a particular packet was dropped. If the nak packet is lost then the retransmit timer will go off and the packet will be retransmitted by normal means. If a senders algorithm continues to count duplicate acks the sender will find itself possibly receiving many duplicate acks after it has already resent the packet due to a nak being received because of the large size of the data pipe. By receiving all of these duplicate acks the sender may find itself doing nothing but resending the same packet of data unnecessarily while keeping the send window closed for absolutely no reason. By removing this feature of the implementation a user can expect to find a satellite connection working much better in the face of errors and other connections should not see any performance loss, but a slight improvement in performance if anything at all.

5. Conclusion

This paper has described two new options that if used will make TCP a more efficient protocol in the face of errors and a more efficient protocol over networks that have a high bandwidth*delay product without decreasing performance over more common networks. If a system that implements the options talks with one that does not, the two systems should still be able to communicate with no problems. This assumes that the system doesn’t use the option numbers defined in this paper in some other way or doesn’t panic when faced with an option that the machine does not implement. Currently at NASA, there are many machines that do not implement either option and communicate just fine with the systems that do implement them.

The drive for implementing big windows has been the direct result of trying to make TCP more efficient over large delay networks [2,3,4,5] such as a T1 satellite. However, another practical use of large windows is becoming more apparent as the local area networks being developed are becoming faster and supporting much larger MTU’s. Hyperchannel, for instances, has been stated to be able to support 1 Mega bit MTU’s in their new line of products. With the current implementation of TCP, efficient use of hyperchannel is not utilized as it should because the physical mediums MTU is larger than the maximum window of the protocol being used. By increasing the TCP

window size, better utilization of networks like hyperchannel will be gained instantly because the sender can send 64 Kbyte packets (IP limitation) but not have to operate in a stop and wait fashion. Future work is being started to increase the IP maximum datagram size so that even better utilization of fast local area networks will be seen by having the TCP/IP protocols being able to send large packets over mediums with very large MTUs. This will hopefully, eliminate the network protocol as the bottleneck in data transfers while workstations and workstation file system technology advances even more so, than it already has.

An area of concern when using the big window mechanism is the use of machine resources. When running over a satellite and a packet is dropped such that 2Z (where Z is the round trip delay) worth of data is unacknowledged, both ends of the connection need to be able to buffer the data using machine mbufs (or whatever mechanism the machine uses), usually a valuable and scarce commodity. If the window size is not chosen properly, some machines will crash when the memory is all used up, or it will keep other parts of the system from running. Thus, setting the window to some fairly large arbitrary number is not a good idea, especially on a general purpose machine where many users log on at any time. What is currently being engineered at NASA is the ability for certain programs to use the setsockopt feature or 4.3bsd asking to use big windows such that the average user may not have access to the large windows, thus limiting the use of big windows to applications that absolutely need them and to protect a valuable system resource.

6. References

[1] Jacobson, V., "Congestion Avoidance and Control", SIGCOMM 88, Stanford, Ca., August 1988.

[2] Jacobson, V., and R. Braden, "TCP Extensions for Long-Delay Paths", LBL, USC/Information Sciences Institute, RFC 1072, October 1988.

[3] Cheriton, D., "VMTP: Versatile Message Transaction Protocol", RFC 1045, Stanford University, February 1988.

[4] Clark, D., M. Lambert, and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol", RFC 998, MIT, March 1987.

[5] Fox, R., "Draft of Proposed Solution for High Delay Circuit File Transfer", GE/NAS Internal Document, March 1988.

[6] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

[7] Leiner, B., "Critical Issues in High Bandwidth Networking", RFC 1077, DARPA, November 1989.

7. Appendix

Both options have been implemented and tested. Contained in this section is some performance gathered to support the use of these two options. The satellite channel used was a 1.544 Mbit link with a 580ms round trip delay. All values are given as units of bytes.

TCP with Big Windows, No Naks:

	transfer rates		
Window Size	no error	10e-7 error rate	10e-6 error rate
64K	94K	53K	14K
72K	106K	51K	15K
80K	115K	42K	14K
92K	115K	43K	14K
100K	135K	66K	15K
112K	126K	53K	17K
124K	154K	45K	14K
136K	160K	66K	15K
156K	167K	45K	14K

Figure 1.

TCP with Big Windows, and Naks:

	transfer rates		
Window Size	no error	10e-7 error rate	10e-6 error rate
64K	95K	83K	43K
72K	104K	87K	49K
80K	117K	96K	62K
92K	124K	119K	39K
100K	140K	124K	35K
112K	151K	126K	53K
124K	160K	140K	36K
136K	167K	148K	38K
156K	167K	160K	38K

Figure 2.

With a 10e-6 error rate, many naks as well as data packets were dropped, causing the wild swing in transfer times. Also, please note that the machines used are SGI Iris 2500 Turbos with the 3.6 OS with the new TCP enhancements. The performance associated with the Irises are slower than a Sun 3/260, but due to some source code restrictions the Iris was used. Initial results on the Sun showed slightly higher performance and less variance.

Author's Address

Richard Fox
950 Linden #208
Sunnyvale, Cal, 94086

EMail: rfox@tandem.com