TCP Reno, New Reno & SACK: A Comparative Analysis

ECE 407

Introduction to Computer Communications

Dr. Do Young Eun

David Bajak

04-27-2007

Congestion control is necessary for networks and the internet in order to provide optimal end to end data communication over an existing variety of communication links. Using congestion control we can utilize network bandwidth as it varies with an activity of users and transmissions by adjusting rate of transmission, and in turn assuring fair network traffic. Creating fair traffic is thus done dynamically by adjusting sender side transmission rates against network congestion. The sender's transmission rate is thus adjusted by first perceiving network congestion, calculating round trip times (rtt) and reception of sent data acknowledgements (ack), then adjusting the sender's transmission rate accordingly. Effectively adjusting transmission rate prevents loss/drops of transmitted data at the receive end which can't be used due to overloading. By transmitting at a rate which the receiver can handle the data prevents further congestion and establishes fair traffic.

TCP utilizes four algorithms which are used to control end-to-end congestion over a network. These algorithms for congestion control include slow start, congestion avoidance, fast retransmit and fast recovery. The slow start algorithm was first implemented by the TCP variant "Reno" and operates on perceiving network congestion by dynamically calculating a value for a congestion window (cwnd) (RFC 2581). Cwnd translates to the sender's transmission rate and is initially set to be less than or equal to 2*SMSS and not be more than 2 segments (RFC 2581 p4). The value of cwnd after initialization is then dynamically calculated against successive acknowledgements for data sent from the receive end. In slow start, cwnd is increased additively by, at most, 1 SMSS until either its value reaches/exceeds the slow start threshold (ssthresh) or a loss of data occurs, which is signified by a missed acknowledgement and translating to congestion somewhere in the network. In either case this results in the end of the current slow start phase (RFC 2581 p4). When the value of cwnd is greater than ssthresh, the system uses a congestion avoidance algorithm until congestion is observed and a loss occurs (RFC 2581 p4).

In the congestion avoidance algorithm the value of cwnd continues to increase additively for each successive ack by no more than the larger of SMSS*SMSS/cwnd or 1 SMSS (RFC 2581 p5) and continues to do so until a loss event occurs. In the event of a loss the congestion avoidance algorithm sets the value of ssthresh to the larger of flightsize/2 (flight size is the amount of data sent and not acknowledged) or 2*SMSS (RFC 2581 p5), resets the value of cwnd to at most 1 SMSS, and puts the system back into the slow start phase, ending the congestion avoidance phase.

To further control congestion, the fast retransmit algorithm is capable of detecting and repairing loss of a segment. In fast retransmit, duplicate acknowledgements are sent to the sender for each segment received out of order and duplicate acknowledgements continue to be sent until the lost packet has been received (RFC 2581 p7). When three of these "duplicate acknowledgments (4 identical acknowledgments without the arrival of any other intervening packets)" (RFC 2581 p6) arrive at the sender it indicates a lost segment. In this case, a segment transmitted has not yet been received or acknowledged by the receiver.

However, following segments have arrived successfully and have been acknowledged by these duplicate acknowledgements. TCP in this case retransmits the un-acknowledged segment because it has most likely been lost or severely delayed and should be retransmitted.

Following the fast retransmit of the perceived missing segment the fast recovery algorithm governs the transmission of new data until a non duplicate acknowledgement is received by the sender (RFC 2581 p6), indicating the lost segment has been received. The fast recovery algorithm first decreases the transmission rate by setting ssthresh to the max of either "flighsize/2 or 2*SMSS" (RFC 2581 p7), (similar to congestion avoidance). It then sets the value of cwnd to "ssthresh + 3SMSS" (RFC 2581 p7), a process termed "inflating" which inflates the congestion window 1 SMSS for each of the 3 following packets (stored in the receivers buffer) that generated duplicate acknowledgements and also 1 SMSS for any other duplicate acknowledgements. This ensures the window size correctly reflects the amount of successfully transmitted segments. The sender can then transmit the next segment with the new window size (RFC 2581 p7), preserving sequential transmission with the new segment. When the sender gets an acknowledgement from the receiver that new data has been received (meaning the lost packet has been received) "cwnd is set to ssthresh" (RFC 2581 p7). This acknowledgement is also an acknowledgement for all following segments (RFC 2581p7); which were stored in the receiver's buffer while the receiver was waiting on the lost segment (RFC 2581 p7).

The Reno TCP variant is defined through the usage of these four congestion control algorithms. The slow start, congestion control, fast retransmit and fast recovery algorithms, developed by Van Jacobson (RFC 2581 p10) as outlined above, specify the TCP internet standard protocol Reno. TCP-Reno, however, is not a flawless protocol and has since been modified and extended through newer TCP variant releases. Reno has limitations regarding to problems with restarting idle connections, generating acknowledgements and as well as loss recovery.

The first problem with TCP-Reno regards TCP connections which have been "idle for a relatively long period of time" (RFC 2581 p7), thus possibly allowing a "potentially inappropriate burst of traffic to be transmitted" (RFC 2581 p7). As a result of the idle TCP connection, "TCP cannot use the acknowledgement clock to strobe new segments into the network, as all of the acknowledgements have drained from the network" (RFC 2581 p7) and can therefore "send a cwnd-size line-rate burst into the network" (RFC 2581 p7). A solution recommended to prevent the traffic burst involves using "slow start to restart transmission after a relatively long idle period" (RFC 2581 p7). In this solution, "when TCP has not received a segment for more than one retransmission timeout, cwnd is reduced to the value of the restart window (RW = IW) before transmission begins" (RFC 2581 p7). This allows the acknowledgement clock to be restarted and probing for bandwidth to begin anew. The traffic burst following idle TCP connections is a "common case of persistent HTTP connections" (RFC 2581 p8), where a “WWW server receives a request before transmitting data to the WWW browser” (RFC 2581 p8), causing reception of the request fail a test for an idle connection and allowing TCP to begin transmission with a large window (RFC 2581 p8).

Another limitation to TCP-Reno regards delays in acknowledgement generation which can affect loss recovery algorithms. Solutions suggest that “the delayed acknowledgment algorithm… should be used by a TCP receiver” (RFC 2581 p8). Ideally “an acknowledgement should be generated for at least every second full-sized segment, and must be generated within 500ms of the arrival of the first unacknowledged packet” (RFC 2581 p8) due to “possible performance problems with generating acknowledgements less frequently than every second full-sized segment” (RFC 2581 p8). Furthermore any “out-of-order data segments should be acknowledged immediately, in order to accelerate loss recovery” (RFC 2581 p8) as well as “send an immediate acknowledgement when (the receiver) receives a data segment that fills in all or part of a gap in the sequence space” (RFC 2581 p8).

There are also limitations in the fast recover and fast retransmit algorithms which are “known to generally not recover very efficiently from multiple losses in a single flight of packets” (RFC 2581 p7). This creates confusion to the TCP sender when identifying which packets to send out because “there is little information available to the TCP sender in making retransmission decisions during Fast Recovery” (RFC 2582 p2). One particular solution to this issue is found through the use of “partial acknowledgements (acknowledgements which cover new data, but not all the data outstanding when loss was detected) to trigger retransmissions” (RFC 2581 p9). The use of partial acknowledgements as an enhancement to fast retransmit/fast recovery and an extension to TCP-Reno “are implicitly allowed, as long as they follow the general principles of the basic four algorithms” (RFC 2581 p9).

Finally, the TCP-Reno variant is vulnerable to attackers which can “impair the performance of a TCP connection by either causing data packets or their acknowledgments to be lost, or by forging excessive duplicate acknowledgments” (RFC 2581 p10). This allows the attacker to drastically reduce transmission rates by “causing two congestion control events back-to-back… often (cutting) ssthresh to its minimum value of 2*SMSS, causing (a) connection to immediately enter the slower-performing congestion avoidance phase” (RFC 2581 p10). This security vulnerability compromises network stability and can “drive a portion of the network into congestion collapse” due to an attacker. In turn, this causes “TCP endpoints to respond more aggressively in the face of congestion by forging excessive duplicate acknowledgments or excessive acknowledgments for new data” (RFC 2581 p10). This security vulnerability however has not been modified in either the New-Reno or SACK extensions of the Reno standard.

Another TCP variant, known as New-Reno and proposed by Janey Hoe, specifically outlines an algorithm for responding to partial acknowledgments (RFC 2582 p2). Using partial acknowledgments improves TCP Reno's fast recovery/fast retransmit limitation when multiple losses occur in a single flight of packets. In this case, this occurs when two or more packets are lost in the flight a partial acknowledgment will be able to "acknowledge some but not all of the packets transmitted before the Fast Retransmit" (RFC 2582 p2). The partial acknowledgments here make it possible to acknowledge portions of data and can prevent confusion when one of the lost packets is received out of order. This modification to the Reno variant is not claimed to be "an optimal modification of Fast Recovery for responding to partial acknowledgements" (RFC 2582 p2), however based on simulations of the algorithm "this modification improves the performance of the Fast Retransmit and Fast Recovery algorithms in a wide variety of scenarios" (RFC 2582 p3).

The changes to the fast retransmit and fast recovery algorithms which define New-Reno are as follows. "When the third duplicate acknowledgement is received and the sender is not already in the Fast Recovery procedure, set ssthresh to no more than… max(FlighSize/2, 2*MSS) (and) record the highest sequence number transmitted in the variable 'recover' " (RFC 2582 p3). The following steps are the same as TCP Reno until the response to partial or new acknowledgments. In New-Reno "when an acknowledgment arrives that acknowledges new data, this acknowledgment could be the acknowledgment elicited by the retransmission from step 2, or elicited by a later retransmission" (RFC 2582 p3). "If this acknowledgement acknowledges all of the data up to and including 'recover', then the acknowledgment acknowledges all the intermediate segments sent between the original transmission of the lost segment and the receipt of the third duplicate acknowledgment (and) set cwnd to either (1) min (ssthresh, FlightSize+MSS) or (2) ssthresh, where ssthresh is the value set in step 1" (RFC 2582 p4) (FlightSize here refers to amount of data outstanding in this step, when fast recovery is exited) and end the fast recovery procedure. However, "if this acknowledgement does not acknowledge all of the data up to and including 'recover', then this is a partial acknowledgement. In this case, retransmit the first unacknowledged segment. Deflate the congestion window by the amount of new data acknowledged, then add back one MSS and send a new segment if permitted by the new value of cwnd" (RFC 2582 p4). The fast recovery procedure should not exit thereafter and "if any duplicate acknowledgements subsequently arrive, execute steps 3 and 4" (RFC 2582 p4). The retransmit timer is also reset "for the first partial acknowledgement arriving during Fast Recovery" (RFC 2582 p4).

The New-Reno variant is not perfect and is limited because "duplicate acknowledgements carry no information to identify the data packet or packets at the TCP data receiver that triggered that duplicate acknowledgement" (RFC 2582 p6). As a result "the TCP data sender is unable to distinguish between a duplicate acknowledgement that results from a lost or delayed data packet, and a duplicate acknowledgement that results from the sender's retransmission of a data packet that had already been received at the TCP data receiver" (RFC 2582 p6). It is possible then that "unnecessary multiple Fast

Retransmits (and multiple reductions of the congestion window)" (RFC 2582 p6) can occur when multiple segments are lost from a single window of data. However "performance problems caused by multiple Fast Retransmits are relatively minor (in Reno and New-Reno compared to Tahoe)" (RFC 2582 p6). In New-Reno "the problem of multiple Fast Retransmits from a single window of data can only occur after a Retransmit Timeout" (RFC 2582 p6). However there is a solution to this problem (bugfix) which involves recording a new variable 'send_high' that can indicate to the receiver and distinguish whether the "sender has unnecessarily retransmitted at least three packets" (RFC 2582 p7). The bugfix to New-Reno is not a necessary adjustment because it is believed that "the differences between any two variants of New-Reno are small compared to the differences between Reno and New-Reno" (RFC 2582 p9). More importantly if "the issue of recovery from multiple dropped packets from a single window of data is of particular importance, the best alternative would be to use the SACK (selective acknowledgement) option" (RFC 2582 p5) described below.

The use of selective acknowledgements (SACK) also extensively improves the TCP Reno variant. With selective acknowledgements "the TCP sender has the information to make intelligent decisions about which packets to retransmit and which packets not to retransmit during Fast Recovery" (RFC 2582 p2). Thus with selective acknowledgements "combined with a selective repeat retransmission policy, can help to overcome these limitations (TCP Reno & New-Reno's multiple lost packets in a single window)" (RFC 2018 p1). With selective acknowledgments the receiver "sends back SACK packets to the sender informing the sender of data that has been received" (RFC 2018 p1) and the sender only retransmits missing segments. Selective acknowledgements are better than cumulative acknowledgements of TCP Reno and New-Reno because cumulative acknowledgments only acknowledge segments that are at the left edge of the receive window and the sender must wait a roundtrip time to find out about each lost packet, which can sometimes lead to unnecessary retransmission of segments which have been correctly received (RFC 2018 p2). This problem is corrected with selective acknowledgements since "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" (RFC 2018 p2).

Selective acknowledgments use two TCP options, "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"(RFC 2018 p2) and "the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted" (RFC 2018 p2). The SACK-permitted option is a "two-byte option which may be sent in a SYN by a TCP that has been extended to receive (and presumably process) the SACK option once the connection has opened" (RFC 2018 p3). The SACK option has variable length and is used "to convey extended acknowledgment information from the receiver to the sender over an established TCP connection" (RFC 2018 p3) and "is to be sent by a data receiver to inform the data sender of non-contiguous blocks of data that have been received and queued" (RFC 2018 p4). When a missing segment is received, "the data receiver acknowledges the data normally by advancing the left window edge in the Acknowledgement Number Field of the TCP header" (RFC 2018 p4). "This option contains a list of some of the blocks of contiguous sequence space occupied by data that has been received and queued within the window" (RFC 2018 p4) where each "block of data queued at the data receiver is defined in the SACK option by two 32-bit unsigned integers in network byte order" (RFC 2018 p4). "The receiver should send an acknowledgment for every valid segment that arrives containing new data and each of these duplicate acknowledgments should bear a SACK option" (RFC 2018 p5). If the receiver chooses to send a SACK "the first SACK block must specify the contiguous block of data containing the segment which triggered this acknowledgement, unless that segment advanced the Acknowledgment Number field in the header" (RFC 2018 p5). The data receiver should also "include as many distinct SACK blocks as possible in the SACK option" (RFC 2018 p5), though there may not be sufficient space to report all blocks present in the receiver's queue (RFC 2018 p5). "The SACK option should be filled out by repeating the most recently reported SACK blocks that are not subsets of a SACK block already included in the SACK option being constructed… after the first SACK block, the following SACK blocks in the SACK option may be listed in arbitrary order" (RFC 2018 p5).

Although selective acknowledgments perform recovery better than cumulative and partial acknowledgments they also have limitations and can also be capable of retransmitting unnecessary packets the receiver has queued. Further, because the SACK option is advisory the receiver is also "permitted to later discard data which (has) been reported in a SACK option" (RFC 2018 p4). "The deployment of other TCP options may reduce the number of available SACK blocks to 2 or even 1" (RFC 2018 p8), which reduce redundancy of SACK delivery when acknowledgements are lost. Another limitation with SACK is that there is a "deployment problem because it requires changes in both the sender and receiver protocol suites" (Influence p327). The SACK option also uses "time-outs as a fall-back mechanism for detecting dropped packets" (RFC 2018 p7) and "when a retransmit timeout occurs the data sender must ignore prior SACK information in determining which data to retransmit" (RFC 2018 p7). Therefore selective acknowledgments and partial acknowledgements both act similarly when time-outs occur, restarting the connection back in the slow-start phase. Such a scenario has excessive consequences when "packet loss is not necessarily an indication of congestion" (RFC 2018 p7) such as wireless and satellite connections.

These TCP variants were "designed for use on wireline links with almost zero link error, (TCP) considers all packet losses to be caused by network congestion" (Flow p341), therefore over wireless networks where link errors exist TCP would incorrectly engage congestion control. These TCP variants are not ideal over wireless networks due to “inherently high link errors, low bandwidth and large delay” (Flow p341). Timeouts can result due to large delays which “may cause RTO to expire if the estimate is not properly set” (Flow p342) and can produce undesirable connection effects if timeouts are handled improperly, resulting in more congestion as well as severely reducing transfer rates. “Link errors cause packets under transmission (to become) corrupted” (Flow p341) resulting in dropped packets and also reducing transfer rate in non-congested networks (Flow p341). Wireless links also have low bandwidth and therefore “should be free from extra overheads as much as possible” (Flow p342).

In order to correct these limitations to TCP it must be extended to work seamlessly over wireless networks. I would first address the problem due to link errors, which “may cause the RTO to expire if the estimate is not properly set” (Flow p342). To do so I would use TCP’s timestamp to accurately measure RTO. Following this TCP would need to be able to distinguish between congestion and link loss. This could be done using the Eifel algorithm where “upon detecting a segment loss, TCP transmission rate is first reduced like normal TCP. Then TCP timestamp value is used to differentiate between whether an acknowledgement has been caused by the original transmitted packet or a retransmitted packet. If the acknowledgement packet corresponds to the original packet, TCP transmission rate would be restored to the previous value” (Flow p344). Also because wireless links have low bandwidth I would attempt to keep the overhead low in order to preserve bandwidth, which can increase the amount of connections.

Works Cited

1. (RFC 2581) - TCP Congestion Control. April 1999 The Internet Society (1999)
2. (RFC 2582) – The NewReno Modification to TCP's Fast Recovery Algorithm. April 1999 The Internet Society (1999)
3. (RFC 2018) – TCP Selective Acknowledgment Options. October 1996 The Internet Society (1996)
4. (Influence) - The Influence of the Large Bandwith-Delay Product on TCP Reno, NewReno, and SACK. Haewon Lee, Soo-hyeong Lee, and Yanghee Choi. School of Computer Science and Engineering Seoul National University. IEEE 2001
5. (Flow) - An Efficient Flow Control Approach For TCP Over Wireless Networks. Salahuddin Muhammad Salim Zabir, Ahmed Ashir, and Norio Shiratori. Journal of Circuits, Systems, and Computers Vol. 13 No. 2 (2004) p341-346. World Scientific Publishing Company 2004