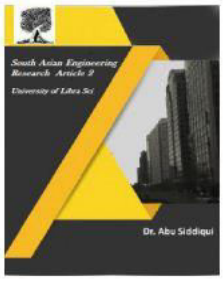
2581-4575

International Journal For Recent Developments in Science & Technology

Crossref

A Peer Reviewed Research Journal



A PROPOSAL OF TCP TO CONTROL HOST TO HOST CONGESTION

[1]SIRAGARAPALLI N S RATNA KONDA BABU , [2]K.N.VENKATESWARA RAO

[1]Department of MCA, SKBR PG College, Amalapuram

[2]Assistant Professor, Department of MCA, SKBR PG College, Amalapuram

ABSTRACT: The Transmission Control Protocol (TCP) carries most Internet traffic, so performance of the Internet depends to a great extent on how well TCP works. Performance characteristics of a particular version of TCP are defined by the congestion control algorithm it employs. This paper presents a survey of various congestion control proposals that preserve the original host-to-host idea of TCP—namely, that neither sender nor receiver relies on any explicit notification from the network. The proposed solutions focus on a variety of problems, starting with the basic problem of eliminating the phenomenon of congestion collapse, and also include the problems of effectively using the available network resources in different types of environments (wired, wireless, high-speed, long-delay, etc.). In a shared, highly distributed, and heterogeneous environment such as the Internet, effective network use depends not only on how well a single TCPbased application can utilize the network capacity, but also on how well it cooperates with other applications transmitting data through the same network. Our survey shows that over the last 20 years many host-to-host techniques have been developed that address several problems with different levels of reliability and precision. There have been enhancements allowing senders to detect fast packet losses and route changes. Other techniques have the ability to estimate the loss rate, the bottleneck buffer size, and level of congestion. The survey describes each congestion control alternative, its strengths and its weaknesses. Additionally, techniques that are in common use or available for testing are described.

KEY WORDS: TCP, congestion control, congestion collapse, packet reordering in TCP, wireless TCP, high-speed TCP.

I.INTRODUCTION

Cost current Internet applications rely on the Transmission Control Protocol (TCP) [1] to deliver data reliably across the network. Although it was not part of its initial design, the most essential element of TCP is congestion control; it defines TCP's performance characteristics. In this paper we present a survey of the congestion control proposals for TCP that preserve its fundamental host-to-host principle, meaning they do not rely on any kind of explicit signaling from the network.1 The proposed algorithms introduce a wide variety of techniques that allow senders to detect loss events, congestion state, and route changes, as well as measure the loss rate, the RTT, the RTT variation, bottleneck buffer sizes, and congestion level with different levels of reliability and precision. The key feature of TCP is its ability to provide a reliable, bi-directional, virtual channel between any two hosts on the Internet. Since the protocol works

2581-4575
International Journal For Recent Developments in Science & Technology
Crossref
A Peer Reviewed Research Journal

over the IP network [3], which provides only best-effort service for delivering packets across the network, the TCP standard [1] specifies a sliding window based flow control.

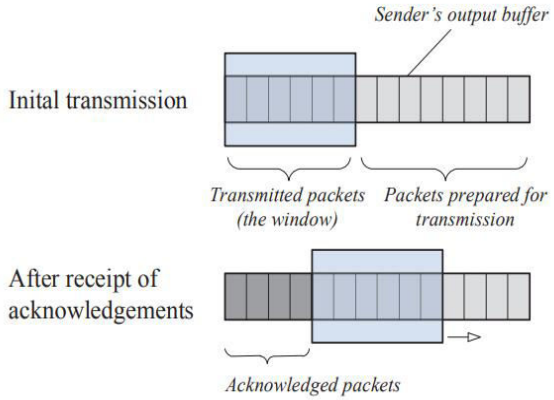


Fig. 1. Sliding window concept: the window "slides" along the sender's output buffer as soon as the receiver acknowledges delivery of at least one packet

This flow control has several mechanisms. First, the sender buffers all data before the transmission, assigning a sequence number to each buffered byte. Continuous blocks of the buffered data are packetized into TCP packets that include a sequence number of the first data byte in the packet. Second, a portion (window) of the prepared packets is transmitted to the receiver using the IP protocol. As soon as the sender receives delivery confirmation for at least one data packet, it transmits a new portion of packets (the window "slides" along the sender's buffer, Figure 1).

Finally, the sender holds responsibility for a data block until the receiver explicitly confirms delivery of the block. As a result, the sender may eventually decide that a particular unacknowledged data block has been lost and start recovery procedures (e.g., retransmit one or several packets). To acknowledge data delivery, the receiver forms an ACK packet that carries one sequence number and (optionally) several pairs of sequence numbers. The former, a cumulative ACK, indicates that all data blocks having smaller sequence numbers have already been delivered. The latter, a selective ACK (Section II-E—a TCP extension standardized 15 years after the introduction of TCP itself), explicitly indicates the ranges of sequence numbers of delivered data packets.

To be more precise, TCP does not have a separate ACK packet, but rather uses flags and option fields in the common TCP header for acknowledgment purposes. (A TCP packet can be both a data packet and an ACK packet at the same time.) However, without loss of generality, we will discuss a notion of ACK packets as a separate entity. Although a sliding window based flow control is relatively simple, it has several conflicting objectives. For example, on the one hand, throughput of a TCP flow should be maximized. This essentially requires that the size of a sliding window also be maximized. (It can be shown that the maximumthroughput of a TCP flow depends directly on the sliding window size and inversely on the round-trip time of the network path.) On the other hand, if the sliding window is too large, there is a high probability of packet loss because the network and the receiver have resource limitations. Thus, minimization of packet losses requires minimizing the sliding window. Therefore, the problem is finding an optimal value for the sliding window (which is usually referred to as the congestion window) that provides good throughput, yet does not overwhelm the network and the receiver.

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal

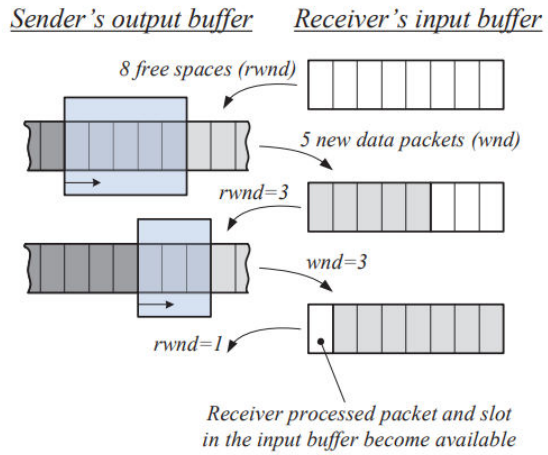


Fig. 2. Receiver's window concept: receiver reports a size of the available input buffer (receiver's window, rwnd) and sender sends a portion (window, wnd) of data packets that does not exceed rwnd

Additionally, TCP should be able to recover from packet losses in a timely fashion. This means that the shorter the interval between packet transmission and loss detection, the faster TCP can recover. However, this interval cannot be too short, or otherwise the sender may detect a loss prematurely and retransmit the corresponding packet unnecessarily. This overreaction simply wastes network resources and may induce high congestion in the network. In other words, when and how a sender detects packet losses is another hard problem for TCP. The initial TCP specification [1] is designed to guard only against overflowing the input buffers at the receiver end. The incorporated mechanism is based on the receiver's window concept, which is essentially a way for the receiver to share the information about the available input buffer with the sender.

Figure 2 illustrates this concept in schematic fashion. When establishing a connection, the receiver informs the sender about the available buffer size for incoming packets (in the example shown, the receiver's window reported initially is 8). The sender transmits a portion (window) of prepared data packets. This portion must not exceed the receiver's window and may be smaller if the sender is not willing (or ready) to send a larger portion. In the case where the receiver is unable to process data as fast as the sender generates it, the receiver reports decreasing values of the window (3 and 1 in the example). This induces the sender to shrink the sliding window. As a result, the whole transmission will eventually synchronize with the receiver's processing rate. Unfortunately, protocol standards that remain unaware of the network resources have created various unexpected effects on the Internet, including the appearance of congestion collapse (see Section II). The problem of congestion control, meaning intelligent (i.e., network resource-aware) and yet effective use of resources available in packet-switched networks, is not a trivial problem, but the efficient solution to it is highly desirable. As a result, congestion control is one of the extensively studied areas in the Internet research conducted over the last 20 years, and a number of proposals aimed at improving various aspects of the congestion-responsive data flows is very large.

Several groups of these proposals have been studied by Hanbali et al. [4] (congestion control in ad hoc networks), Lochert et al. [2] (congestion control for mobile ad hoc networks), Widmer et al. [5] (congestion control for non-TCP protocols), Balakrishnan et al. [6] (congestion control for wireless networks), Leung at al. [7] (congestion control

2581-4575
International Journal For Recent Developments in Science & Technology
Crossref
A Peer Reviewed Research Journal

for networks with high levels of packet reordering), Low et al. [8] (current up to 2002 TCP variants and their analytical models), Hasegawa and Murata [9] (fairness issues in congestion control), and others researchers. Unlike previous studies, in this survey we tried to collect, classify, and analyze major congestion control algorithms that optimize various parameters of TCP data transfer without relying on any explicit notifications from the network. In other words, they preserve the host-to-host principle of TCP, whereby the network is seen as a black box.

II. CONGESTION COLLAPSE

The initial TCP standard has a serious drawback: it lacks any means to adjust the transmission rate to the state of the network. When there are many users and user demands for shared network resources, the aggregate rate of all TCP senders sharing the same network can easily exceed (and in practice do exceed) the capacity of the network. It is commonly known in the flow-control world that if the offered load in an uncontrolled distributed sharing system (e.g., road traffic) exceeds the total system capacity, the effective load will go to zero (collapses) as load increases [10] (Figure 3). With regard to TCP, the origins of this effect, known as a congestion collapse [11]–[13], can be illustrated using a simple example. Let us consider a router placed somewhere between networks A and B which generate excessive amounts of TCP traffic (Figure 4). Clearly, if the path from A to B is congested by 400% (4 times more than the router can deliver), at least 75% of all packets from network A will be dropped and at most 25% of data packets may result in ACKs. If the reverse path from B to A is also congested (also by 400%, for example), the chance that ACK packets get through is also 25%. In other words, only 25% of 25% (i.e., 6.25%) of the data packets sent from A to B will be acknowledged successfully. If we assume that each data packet requires its own acknowledgement (not a requirement for TCP, but serves to illustrate the point), then a 75% loss in each direction causes a 93.75% drop in throughput (goodput) of the TCPlike flow.

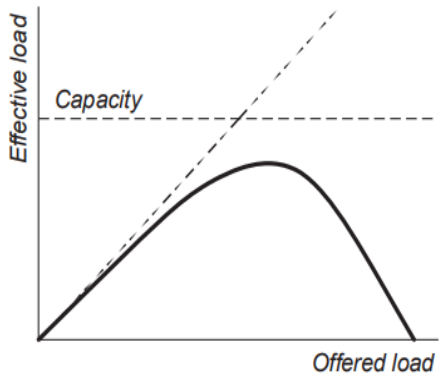


Fig. 3. Effective TCP load versus offered load from TCP senders

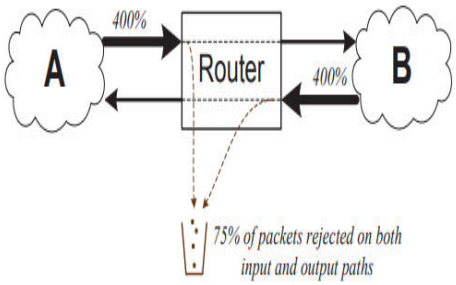


Fig. 4. Congestion collapse rationale. 75% of data packets dropped on forward path and 75% of ACKs dropped on reverse: only 6.25% of packets are acknowledged

To resolve the congestion collapse problem, a number of solutions have been proposed. All of them share the sameidea, namely of introducing

2581-4575
International Journal For Recent Developments in Science & Technology
Crossref
A Peer Reviewed Research Journal

a network-aware rate limiting mechanism alongside the receiver-driven flow control. For this purpose the congestion window concept was introduced: a TCP sender's estimate of the number of data packets the network can accept for delivery without becoming congested. In the special case where the flow control limit (the socalled receiver window) is less than the congestion control limit (i.e., the congestion window), the former is considered a real bound for outstanding data packets. Although this is a formal definition of the real TCP rate bound, we will only consider the congestion window as a rate limiting factor, assuming that in most cases the processing rate of end-hosts is several orders of magnitude higher than the data transfer rate that the network can potentially offer.

Additionally, we will compare different algorithms, focusing on the congestion window dynamics as a measure of the particular congestion control algorithm effectiveness. In the next section we will discuss basic congestion control algorithms that have been proposed to extend the TCP specification. As we shall see, these algorithms not only preserve the idea of treating the network as a black box but also provide a good precision level to detect congestion and prevent collapse. Table I gives a summary of features of the various algorithms. Additionally, Figure 5 shows the evolutionary graph of these algorithms. However, solving the congestion problem introduces new problems that lead to network channel underutilization. Here we focus primarily on the congestion problem itself and basic approaches to improve data transfer effectiveness. In the following sections other problems and solutions will be discussed.

III. HOST TO HOST CONGESTION FOR TCP

A.TCP Tahoe

One of the earliest host-to-host solutions to solve the congestion problem in TCP flows has been proposed by Jacobson [14]. The solution is based on the original TCP specification (RFC 793 [1]) and includes a number of algorithms that can be divided into three groups. The first group tackles the problem of an erroneous retransmission timeout estimate (RTO). If this value is overestimated, the TCP packet loss detection. mechanism becomes very conservative, and performance of individual flows may severely degrade. In the opposite case, when the value of the RTO is underestimated, the error detection mechanism may perform unnecessary retransmissions, wasting shared network resources and worsening the overall congestion in the network. Since it is practically impossible to distinguish between an ACK for an original and a retransmitted packet, RTO calculation is further complicated. The round-trip variance estimation (rttvar) algorithm tries to mitigate the overestimation problem.

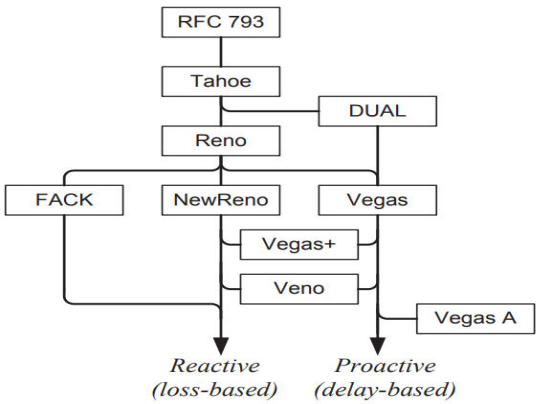
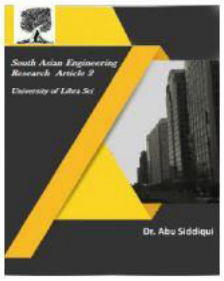


Fig. 5. Evolutionary graph of TCP variants that solve the congestion collapse problem

2581-4575
International Journal For Recent Developments in Science & Technology
Crossref
A Peer Reviewed Research Journal


Instead of a linear relationship between the RTO and estimated round-trip time (RTT) value (β · SRT T , in which β is a constant in range from 1.3 to 2 [1] and SRT T is an exponentially averaged RTT value), the algorithm calculates an RTT variation estimate to establish a fine-grained upper bound for the RTO (SRT T + 4 · rttvar). The exponential retransmit timer backoff algorithm solves the underestimation problem by doubling the RTO value on each retransmission event. In other words, during severe congestion, detection of subsequent packet losses results in exponential RTO growth, significantly reducing the total number of retransmissions and helping stabilize the network state. The ACK ambiguity problem is resolved by Karn's clamped retransmit backoff algorithm [26]. Importantly, the RTT of a data packet that has been retransmitted is not used in calculation for the average RTT and RTT variance, and thus it has no impact on the RTO estimate.

The second group of algorithms enhances the detection of packet losses. The original TCP specification defines the RTO as the only loss detection mechanism. Although it is sufficient to reliably detect all losses, this detection is not fast enough. Clearly, the minimum time when loss can be detected is the RTT—i.e., if the receiver is able to instantly detect and report a loss to the sender, the report will reach the sender exactly one RTT after sending the lost packet. The RTO, by definition, is greater than RTT.

If we require that TCP receivers immediately reply to all out-of-order data packets with reports of the last inorder packet (a duplicate ACK) [27], the loss can be detected by the Fast Retransmit algorithm [28], almost within the RTT interval. In other words, assuming the probability of packet reordering and duplication in the network is negligible, the duplicate ACKs can be considered a reliable loss indicator.

Having this new indicator, the sender can retransmit lost data without waiting for the corresponding RTO event. The third and most important group includes the Slow Start and Congestion Avoidance algorithms. These provide two slightly different distributed host-to-host mechanisms which allow a TCP sender to detect available network resources and adjust the transmission rate of the TCP flow to the detected limits. Assuming the

probability of random packet corruption during transmission is negligible (1%), the sender can treat all detected packet losses as congestion indicators. In addition, the reception of any ACK packet is an indication that the network can accept and deliver at least one new packet (i.e., the ACKed packet has left and a new one can enter the network). Thus the sender, reasonably sure it will not cause congestion, can send at least the amount of data that has just been acknowledged. This in-out packet balancing is called the packet conservation principle and is a core element, both of Slow Start and of Congestion Avoidance

B. TCP DUAL

TCP Tahoe (Section II-A) has rendered a great service to the Internet community by solving the congestion collapse problem. However, this solution has an unpleasant drawback of straining the network with high-amplitude periodic phases.

This behavior induces significant periodic changes in sending rate, round-trip time, and

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal

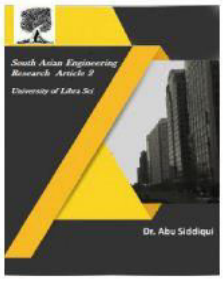
network buffer utilization, leading to variability in packet losses. Wang and Crowcroft [15] presented TCP DUAL, which refines the Congestion Avoidance algorithm. DUAL tries to mitigate the oscillatory patterns in network dynamics by using a proactive congestion detection mechanism coupled with softer reactions to detected events. More specifically, it introduces the queuing delay as a prediction parameter of the network congestion state. Let us assume that routes do not change during the transmission and that the receiver acknowledges each data packet immediately. Then we can consider the minimal RTT value observed by the sender (RT Tmin) as a good indication that the path is in a congestion-free state

C. TCP Reno

Reducing the congestion window to one packet as a reaction to packet loss, as occurs with TCP Tahoe (Section II-A), is rather draconian and can, in some cases, lead to significant throughput degradation. For example, a 1% packet loss rate can cause up to a 75% throughput degradation of a TCP flow running the Tahoe algorithm [16]. To resolve this problem, Jacobson [16] revised the original composition of Slow Start and Congestion Avoidance by introducing the concept of differentiating between major and minor congestion events.

A loss detection through the retransmission timeout indicates that for a certain time interval (as an example, RT O minus RT T) some major congestion event has prevented delivery of any data packets on the network. Therefore, the sender should apply the conservative policy of resetting the congestion window to a minimal value. Quite a different state can be inferred from a loss detected by duplicate ACKs. Suppose the sender has received four ACKs, where the first one acknowledges some new data and the rest are the exact copies of the first one (usually referred to as three duplicate ACKs). The duplicate ACKs indicate that the some packets have failed to arrive. Nonetheless, presence of each ACK—including the duplicates—indicates the successful delivery of a data packet. The sender, in addition to detecting the lost packet, is also observing the ability of the network to deliver some data. Thus, the network state can be considered to be lightly congested, and the reaction to the loss event can be more optimistic. In TCP Reno, the optimistic reaction is to use the Fast Recovery algorithm [17].

The intention of Fast Recovery is to halve a flow's network share (i.e., to halve the congestion window) and to taper back network resource probing (holding all growth in the congestion window) until the error is recovered. In other words, the sender stays in Fast Recovery until it receives a non-duplicate acknowledgment. The algorithm phases are illustrated in Figure 14, where congestion window sizes (cwnd) in various states are denoted as the line segments above the State lines, and the arrows indicate the effective congestion window size— the amount of packets in transit. The transition from State 1 to State 2 shows the core concept of optimistic network share reduction, using the multiplicative decrease policy. After the reduction (i.e., from cwnd to cwnd/2), the algorithm not only retransmits the oldest unacknowledged data packet (i.e., applies the

2581-4575
International Journal For Recent Developments in Science & Technology
Crossref
A Peer Reviewed Research Journal


Fast Retransmit algorithm), but also inflates the congestion window by the number of duplicate packets.

D. TCP NewReno

One of the vulnerabilities of TCP Reno's Fast Recovery algorithm manifests itself when multiple packet losses occur as part of a single congestion event. This significantly decreases Reno's performance in heavy load environments. This problem is demonstrated in Figure 17, where a single congestion event (e.g., a short burst of cross traffic) causes the loss of several data packets (indicated by x). As we can see, the desired optimistic reaction of Fast Recovery (i.e., the congestion window halving) suddenly transforms into a conservative exponential congestion window decrease. That is, the first loss causes entry into the recovery phase and the halving of the congestion window. The reception of any non-duplicate ACK would finish the recovery. However, the subsequent loss detections cause the congestion window to decrease further, using the same mechanisms of entering and exiting the recovery state.

In one sense, this exponential reaction to multiple losses is expected from the congestion control algorithm, the purpose of which is to reduce consumption of network resources in complex congestion situations. But this expectation rests on the assumption that congestion states, as deduced from each detected loss, are independent, and in the example above this does not hold true.

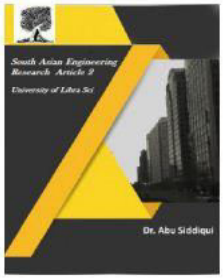
All packet losses from the original data bundle (i.e., from those data packets outstanding at the moment of loss detection) have a high probability of being caused by a single congestion event. Thus, the second and third losses from the example above should be treated only as requests to retransmit data and not as congestion indicators.

Moreover, reducing the congestion window does not guarantee the instant release of network resources. All packets sent before the congestion window reduction are still in transit. Before the new congestion window size becomes effective, we should not apply any additional rate reduction policies.

This can be interpreted as reducing the congestion window no more often than once per one-way propagation delay or approximately RT T /2. Floyd et al. [18], [19] introduce a simple refinement of Reno's Fast Recovery. It solves the ambiguity of congestion events by restricting the exit from the recovery phase until all data packets from the initial congestion window are acknowledged. More formally, the NewReno algorithm adds a special state variable to remember the sequence number of the last data packet sent before entering the Fast Recovery state. This value helps distinguish between partial and new data ACKs.

The reception of a new data ACK means that all packets sent before the error detection were successfully delivered and any new loss would reflect a new congestion event. A partial ACK confirms the recovery from only the first error and indicates more losses in the original bundle of packets. Figure 18 illustrates the differences between Reno and NewReno.

Similar to the original Reno algorithm, reception of any duplicate ACKs triggers only the inflation of the congestion window (States 3, 4, 6). A partial ACK provides the exact information about some part of the delivered

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal


data. Therefore, reaction to partial ACK is only a deflation of the congestion window (State 4) and a retransmission of the next unacknowledged data packet (State 5). Finally, exit from the NewReno's Fast Recovery can proceed only when the sender receives a new data ACK, which is accompanied by the full congestion window deflation.

E. TCP SACK

The TCP specification [1] defines that the only feedback message be in the form of cumulative ACKs, i.e., acknowledgments of only the last in-order delivered data packet. This property limits the ability of the sender to detect more than one packet loss per RTT. For example, if a second and a third data packet from some continuous TCP stream are lost, the receiver, according to the cumulative ACK policy, would reply to fourth and consecutive packets with duplicate acknowledgments of the first packet.

Moreover, because Fast Recovery assumes loss of only one data packet—i.e., only one packet will be retransmitted after a loss detection—loss of the third data packet will be detected after another RTT at best.

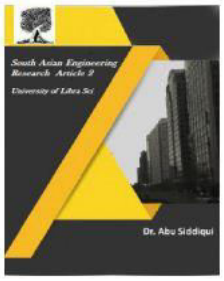
Thus, a duration of the recovery in Reno is directly proportional to the number of packet losses and RTT. NewReno resolves Reno's problem of excessive rate reducing in the presence of multiple losses, but it does not solve the fundamental problem of prolonged recovery. The recovery process can be sped up if the sender retransmits several packets instead of a single one upon error detection. However, this technique assumes certain patterns of packet losses and may just waste network resources if actual losses deviate from these patterns. If a receiver can provide information about several packet losses within a single feedback message, the sender would be able to implement a simple algorithm to resolve the long recovery problem. Moreover, Reno's problem discussed in Section II-D can be solved by restricting the congestion window reduction to no more than once per RTT period, instead ofimplementing the NewReno algorithm. The rationale behind this solution is that in the worst case, the interval between the first and last data packets sent before reception of any ACK is exactly one RTT.

F. TCP FACK

We have informally discussed one possible extension of the Reno algorithm utilizing SACK information, whereby the congestion window is not multiplicatively reduced more than once per RTT. Another approach is the FACK (Forward Acknowledgments) congestion control algorithm [21]. It defines recovery procedures which, unlike the Fast Recovery algorithm of standard TCP (TCP Reno), use additional information available in SACK to handle error recovery (flow control) and the number of outstanding packets (rate control) in two separate mechanisms. The flow control part of the FACK algorithm uses selective ACKs to indicate losses.

It provides a means for timely retransmission of lost data packets, as well.

Because retransmitted data packets are reported as lost for at least one RTT and a loss cannot be instantly recovered, the FACK sender is required to retain information about retransmitted data. This information should at least include the time of the last retransmission in order to detect a loss using the legacy

2581-4575
International Journal For Recent Developments in Science & Technology
Crossref
A Peer Reviewed Research Journal


timeout method (RTO). The rate control part, unlike Reno's and NewReno's Fast Recovery algorithms, has a direct means to calculate the number of outstanding data packets using information extracted from SACKs.

G. TCP Vegas

Brakmo and Peterson [22] proposed the Vegas algorithm as another proactive method to replace the reactive Congestion Avoidance algorithm. The key component is making an estimate of the used buffer size at the bottleneck router. Similar to the DUAL algorithm, this estimate is based on RTT measurements. The minimal RTT value observed during the connection lifetime is considered a baseline measurement indicating a congestion-free network state (analogous to Figure 12).

In other words, a larger RTT is due to increased queuing in the transmission path. Unlike DUAL, Vegas tries to quantify, not a relative, but an absolute number of packets enqueued at the bottleneck router as a function of the expected and actual transmission rate.

If no packets are dropped in the network, Vegas controls the congestion window using an additive increase and additive decrease (AIAD) policy. Reactions to packet losses are defined by any of the standard congestion control algorithms (either Reno, NewReno, or FACK). Additionally, Vegas revises the Slow Start algorithm by slowing-down the opportunistic network resource probing. In particular, the updated algorithm restricts the congestion window to increase every other RTT.

H. TCP Vegas+

Hasegawa et al. [23] have recognized a serious problem in TCP Vegas which prevents any attempts to deploy it. The Vegas proactive congestion-prevention mechanism (limiting buffering in the path) cannot effectively compete with the highly deployed Reno reactive mechanism (inducing network buffering and buffer overflowing).

This point can be illustrated using an idealized convergence diagram for competition between Reno and Vegas flows.

TCP Vegas+ was proposed as a way to provide a way of incremental Vegas deployment.

For this purpose, Vegas+ borrows from both the reactive (Reno-like aggressive) and proactive (Vegas-like moderate) congestion avoidance approaches. More specifically, the Congestion Avoidance phase of Vegas+ initially assumes a Vegas-friendly network environment and employs bottleneck buffer estimation to control the congestion window (i.e., Vegas rules).

At the moment when an internal heuristic detects a Vegas-unfriendly environment, Congestion Avoidance falls back to the Reno algorithm.

The Vegas-friendliness/unfriendliness detection heuristic is based on a trend estimate of the RTT. The special state variable C is increased if the sender estimates an increase in the RTT and concurrently the size of the congestion window is unchanged or even reduced. In the opposite case, if the estimated RTT grows smaller, C is decreased. Clearly, large values of C indicate a Vegas-unfriendly network state (i.e., if the congestion window is stable, the RTT also should be stable).

International Journal For Recent Developments in Science & Technology
2581-4575
Crossref
A Peer Reviewed Research Journal

IV. RESULTS

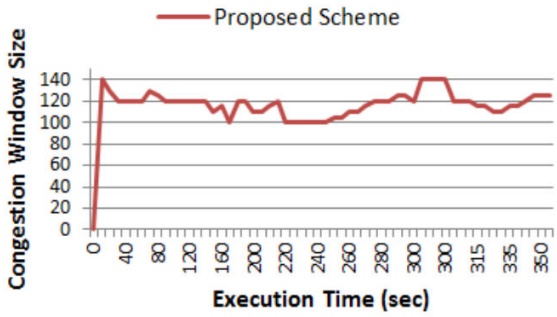


Fig. 6: SIMULATION RESULTS FOR WINDOW SIZE OF THE PROPOSED SCHEME

V. CONCLUSION

In this work we have presented a survey of various approaches to TCP congestion control that do not rely on any explicit signaling from the network. The survey highlighted the fact that the research focus has changed with the development of the Internet, from the basic problem of eliminating the congestion collapse phenomenon to problems of using available network resources effectively in different types of environments (wired, wireless, high-speed, long-delay, etc.). refine the core congestion control principle by making more optimistic assumptions about the network (Reno, NewReno). refine the TCP protocol to include extended reporting abilities of the receiver (SACK, DSACK), which allows the sender to estimate the network state more precisely (FACK, RR-TCP).

VI. REFERENCES

[1] J. Postel, "RFC793—transmission control protocol," RFC, 1981.

[2] C. Lochert, B. Scheuermann, and M. Mauve, "A survey on congestion control for mobile ad hoc networks," Wireless Communications and Mobile Computing, vol. 7, no. 5, p. 655, 2007. [3] J. Postel, "RFC791—Internet Protocol," RFC, 1981.

[4] A. Al Hanbali, E. Altman, and P. Nain, "A survey of TCP over ad hoc networks," IEEE Commun. Surveys Tutorials, vol. 7, no. 3, pp. 22–36, 3rd quarter 2005.

[5] J. Widmer, R. Denda, and M. Mauve, "A survey on TCP-friendly congestion control," IEEE Network, vol. 15, no. 3, pp. 28–37, May/June 2001.

[6] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," IEEE/ACM Trans. Netw., vol. 5, no. 6, pp. 756–769, December 1997.

[7] K.-C. Leung, V. Li, and D. Yang, "An overview of packet reordering in transmission control protocol (TCP): problems, solutions, and challenges," IEEE Trans. Parallel Distrib. Syst., vol. 18, no. 4, pp. 522–535, April 2007.

[8] S. Low, F. Paganini, and J. Doyle, "Internet congestion control," IEEE Control Syst. Mag., vol. 22, no. 1, pp. 28–43, February 2002.

[9] G. Hasegawa and M. Murata, "Survey on fairness issues in TCP congestion control mechanisms," IEICE Trans. Commun. (Special Issue on New Developments on QoS Technologies for Information Networks), vol. E84-B, no. 6, pp. 1461–1472, June 2001.

[10] M. Gerla and L. Kleinrock, "Flow control: a comparative survey," IEEE Trans. Commun., vol. 28, no. 4, pp. 553–574, April 1980