End-to-end congestion control algorithms for the Internet

João Poupino

1 Introduction

The Internet has grown in a significant way in the last twenty years. Today, it connects millions of people and machines, and it has impacted numerous aspects of our lives. Next-generation consumer networks, such as residential fiber optic networks [20], are pushing the boundaries of what was considered to be impossible just a few years ago. For example, new services with ever increasing bandwidth requirements, such as high definition video-on-demand, on-line music, and cloud-based storage services, are quickly changing the way we use the Internet. High-speed transatlantic networks are enabling laboratories, like CERN and Fermilab, to exchange massive amounts of experimental data with science research facilities across the globe everyday.

The Transmission Control Protocol (TCP) [38] plays a critical role in the Internet infrastructure. By providing a connection-oriented, reliable, byte-stream service, TCP has enabled the Internet to thrive on many types of networks; ranging from high-speed, highly reliable fiber optic links to error prone wireless links. Additionally, TCP provides a fundamental service to the stability of the Internet, called congestion control.

Congestion control is a key issue in any network. It is intrinsically related to network and router capacity. The Internet, being composed by many heterogeneous networks with distinct capacities, is particularly susceptible to congestion events. Routers must buffer packets in queues before and after processing. If the arrival rate of packets is larger than the capacity of the network, a router must store them in a queue until it can forward the packets to the next hop. Eventually, if the arrival rate continues to be larger than the departure rate, the queue will grow to a size where the router has no alternative but to drop packets, disrupting network flows. So, as the packet loss rate increases, throughput will decrease. The solution is to decrease router queue occupancy, and consequently packet loss rates. To achieve this, sources must lower their sending rate. However, this was not the behavior of early TCP versions. TCP, being a reliable transport protocol, is expected to retransmit lost segments, but its original retransmission behavior was too aggressive, putting the network at risk of congestion collapse. A congestion collapse happens when a severely congested network reaches a steady state, where the actual useful throughput is very low.

In October of 1986, the Internet suffered its first congestion collapse, and more would soon follow. These events fascinated researchers, prompting them to investigate the underlying causes of the collapses [21]. It was shown that TCP lacked the fundamental control mechanisms to prevent congestion collapses. Several algorithms were proposed, along with critical improvements to TCP timers.

The TCP congestion control algorithms have been successful since their inception in 1988, and have allowed the Internet to scale multiple times. However, the proliferation of new-generation networks, such as links with high bandwith-delay products or wireless networks, have made the limitations of traditional TCP congestion control more apparent. Even with its long history, TCP congestion control remains an hot topic to this day, as TCP is still the only standard Internet protocol capable of end-to-end reliable transmission, and congestion control.

2 Current TCP challenges and solutions

A major challenge currently faced by TCP is link efficiency. Link efficiency means TCP should be able to fully use the capacity provided by a network. In fact, many recent contributions to TCP

congestion control come as a side effect of the need to enhance TCP's performance on links with large delay-bandwidth products. To understand this issue, we note that advances in physical network technology have made long distance, multi-gigabit, networks commonplace. However, although network capacity has been continually increasing, latency remains constant. We also note that a TCP sender adjusts its window according to the Additive Increase Multiplicative Decrease (AIMD) algorithm with fairly conservative parameters. This design, while safe, leads to very poor link utilization on networks with large delay-bandwidth products. This problem is aggravated as the bandwidth-delay product of the link increases.

Fairness is also an important issue. We consider two different types of fairness. TCP fairness, which is how fair a new TCP proposal will be to standard TCP, and intra-protocol fairness, which is concerned with fair distribution of bandwidth among flows running the same TCP version. Regarding legacy TCP fairness, bandwidth allocation for flows running TCP variants should not dominate standard TCP flows, to allow for a less disruptive deployment in the Internet. Regarding intra-protocol fairness, any TCP algorithm which aims to provide efficient link utilization on high-speed networks, must eventually converge to fair distribution of bandwidth among competing flows. Also, the algorithm should account for various aspects that may influence individual flow behavior, and consequently its overall fairness [36].

Wireless networks have become very popular in the last few years. With the advent of mobile computing, the number of devices using TCP over wireless links has seen an explosive growth. They serve as a testimony of TCP's flexibility to adapt to new physical media. However, it is important to note that TCP was originally designed with wired networks in mind, where packet error rates are small, and the network characteristics are fairly constant. This, of course, contrasts with the reality of wireless networks. In wireless networks, error rates can spike due to interference, or frame collisions, caused by multiple senders trying to transmit simultaneously. Available bandwidth can suddenly drop due to signal degradation. TCP is designed to interpret packet loss as a sign of congestion. So, in a wireless network, when TCP detects that a segment is lost, it will frequently and erroneously assume that it was caused by congestion. In response to the loss event, TCP will lower its congestion window by half, unnecessarily affecting throughput. This is, perhaps, the most challenging problem of TCP, as it does not have any means to distinguish between losses caused by network congestion, and losses caused by link errors.

The limitations in TCP congestion control urged the research community to put forward new solutions. Many contributions have been proposed, but all can be framed in one of two approaches: end-to-end congestion control, and router-assisted congestion control.

The end-to-end strategy is the classical approach to congestion control. In this model, it is the responsibility of the TCP sender to detect congestion and act upon it. End-to-end algorithms have a black-box perspective of the network. Therefore, all end-to-end congestion mechanisms can only rely on implicit congestion signals, i.e. packet losses and delay variations. This approach has proved successful for many years. Nevertheless, it has some limitations. Loss-based schemes cause network congestion by design, as it is the only way to probe for available bandwidth, making them reactive, rather than proactive. Delay-based schemes can proactively respond to network congestion, and thus are more network friendly. By inspecting fluctuations in measured RTT values, they can infer router queue occupancy, but face many challenges with accurate RTT estimation and link under-utilization on the presence of loss-based algorithms. Moreover, it is yet to be determined if pure delay-based algorithms can work correctly on many network scenarios. More recently, hybrid-based (i.e., that combine loss and delay) approaches are being proposed as possible solutions to some of the inherent problems with pure loss- and pure delay-based mechanisms. Hybrid-based algorithms show improved results in some scenarios, but they still retain most problems from the loss and delay based algorithms, because they still have limited awareness of the true network state. Despite its limitations, the proponents

of the end-to-end approach argue that only an end-to-end algorithm adheres to the "golden principle" that complexity of a network should be at its edge and not at its core [14], allowing the network to scale [41]. Another important argument is that the end-to-end approach allows for an incremental deployment of such algorithms. A very attractive characteristic of and end-to-end congestion control algorithm is that only the sender side must be modified.

At the same time, a different strategy to congestion control is being proposed. The router-assisted approach makes routers an active component of a congestion control architecture.

One early proposal is Random Early Detection (RED) [16] alongside Explicit Congestion Notification (ECN) [40]. RED aims to mitigate some of the problems with drop-tail queues. In drop tail queues, packets are dropped in a bottleneck router, after a queue buffer overflow occurs. In RED managed queues, packets will be dropped earlier, based on the output of a drop probability algorithm. If the queue occupancy is low, almost no packets are dropped; conversely, if the queue starts to fill, the drop probability will rise proportionally. This approach has shown good results in controlling packet drops, queuing delay and lock-out behavior [6]. However, TCP senders still detect network congestion by observing packet losses, and inherit all the problems associated with implicit congestion signaling. Thus, ECN was proposed. ECN uses the output from RED's probability function, but instead of dropping the packet, it will explicitly signal the TCP sender, allowing it to adapt to congestion before it occurs and avoid unnecessary retransmissions.

More sophisticated algorithms for Internet congestion control, called explicit-rate feedback algorithms, have also been proposed. They were inspired on ATM's ABR protocol [11]. Examples of such proposals are [3] and XCP [12], even though XCP uses windows, instead of rates. In essence, explicit-rate feedback algorithms allow routers to present sources with how much data they should be sending at a given time. Therefore, they exhibit advantages over the end-to-end counterparts, solving most of their problems. This includes efficient utilization in high bandwith-delay links, flow fairness, and the ambiguity of segment loss in heterogeneous networks. The main drawbacks are that, in addition to the sources, all routers in the network are required to participate in the protocol, and that more powerful routers are needed to cope with the increased complexity of such algorithms.

The router-assisted approach, while arguably superior, is still under discussion, since its implementation in the current Internet is very challenging due to both technical and political aspects. This, and other factors, have hindered so far the adoption of the router-assisted approach.

Our work will focus on the end-to-end approach.

3 Related work

End-to-end algorithms propose an attractive approach to Internet congestion control, both in simplicity and scalability. We present some of the most relevant work on TCP end-to-end congestion control. Recall that the only signals of network congestion available to an end-to-end algorithm are packet losses and latency variations. Therefore, research has been focused on three types of algorithms:

- Loss-based algorithms rely on packet losses alone to react to network congestion;
- Delay-based algorithms use delay measurements alone to infer router queue occupancy and act before heavy congestion occurs;
- Hybrid-based algorithms use techniques from both loss-based and delay-based algorithms; their rationale is that with more information, an end-to-end algorithm can infer the state of the network more accurately and make better decisions.

3.1 Loss-based congestion control algorithms

Traditionally, TCP has used packet losses to detect network congestion. Many new algorithms now use delay information, but pure loss-based algorithms are still, by far, dominant in the Internet. We present

the key concepts of the classic TCP congestion control algorithms, and their significance to Internet stability.

Tahoe was the first TCP version designed with congestion control [21]. One of the main principles of its operation is that the rate of new packets sent into the network must be close to the rate at which the receiver returns the acknowledgements. In versions earlier than Tahoe, sources send multiple segments into the network until the advertised receiver window is full. This overly aggressive behavior may cause problems when there are slower links between the sender and receiver. Upon reaching an upstream router connected to a lower capacity link, a packet must be buffered and wait for transmission. Since the buffer space is finite, it is possible that the router overflows its queue, and multiple packets will be dropped. It was shown in [21] how that approach could severely affect the throughput of TCP connections.

Tahoe includes three new algorithms. They are slow-start, congestion avoidance, and fast retransmit. It also adds a new window to the sender's TCP, called the congestion window. Moreover, the TCP sender must not transmit more than the minimum of the congestion window and the advertised receiver window. One of TCP Tahoe's fundamental assumptions is that the packet loss rate is very small, therefore packet losses could be inferred as a strong indicator that congestion had occurred on the network, and the sender should decrease its sending rate. Tahoe, even with its limitations, was a major breakthrough in congestion control. It has played a critical part in the prevention of congestive collapses on the Internet, and has set the guidelines and principles for which virtually every new TCP implementation should respect or, at least, consider.

Tahoe was followed by Reno [2]. Reno retains most of Tahoe's characteristics, but introduces two changes that improve Tahoe's performance considerably. The key idea is that Reno, contrarily to Tahoe, should not fall back to slow-start when packet loss is detected through duplicated ACKs. Instead, it should continue sending data, albeit at a slower pace. The main insight is that since the sender is still receiving ACKs, the network is still delivering packets, and so the congestion experienced is not yet heavy. As such, Reno introduced a new mechanism called called fast recovery. Reno is a major improvement over Tahoe regarding single packet loss in a window. However, like Tahoe, it does not behave well when multiple packet losses occur within a single window of data.

NewReno is proposed in [15] to enhance Reno's fast retransmit and fast recovery algorithms. Reno assumes that only one segment is lost from a window of data. However, packets frequently arrive in bursts at the routers [31], and therefore packet drops will also occur in bursts. In this setting, Reno will not handle multiple packet drops adequately.

A fundamental problem with Reno, which makes it unsuitable to handle multiple packet drops, is that every time an ACK that acknowledges new data arrives, it will leave fast recovery. In a scenario where multiple packets are dropped, Reno will call fast retransmit and fast recovery multiple times. Each time this happens, the congestion window and slow-start threshold will be further reduced, leading to poor throughput.

The novel idea in NewReno is the introduction of partial acknowledgements. A partial acknowledgement will acknowledge only some of the segments before the sender had detected packet loss. In Reno, the first partial acknowledgement will result in the termination of fast recovery. NewReno, on the other hand, will use the partial acknowledgement as a signal that another packet was lost, and react accordingly.

Upon detecting three duplicate acknowledgements, NewReno will save the sequence number of the last transmitted segment in a variable called `recover`. What follows is a behavior nearly identical to Reno. NewReno will enter fast retransmit, send the lost packet, and invoke fast recovery. However, when a partial acknowledgement arrives at the sender, NewReno will retransmit the segment matching the partial acknowledgement, but it will continue in fast recovery. Only after the sequence number in `recover` is acknowledged, will fast recovery end.

NewReno is still nowadays one of the most popular TCP variants. Together with Reno and SACK [35], which extends TCP's capabilities to selectively acknowledge individual segments, and therefore improve its ability to recover from multi-window segment losses, it is commonly regarded as the *de*

facto TCP version.

High-speed TCP congestion control algorithms

It is widely recognised that TCP's conservative AIMD behavior scales poorly in networks with high bandwith-delay products. One possible strategy to improve link efficiency is to make TCP more aggressive. That is, increase the congestion window faster, and upon packet loss, use a smaller decrease factor. Several new high-speed TCP variants were proposed as simple changes to TCP's AIMD mechanism.

HighSpeed TCP for Large Congestion Windows [13] modifies TCP's AIMD mechanism to make it more suitable for high-speed networks. It is designed to have a dynamic behavior that is dependent on the state of the network. In low packet loss conditions, HighSpeed TCP behaves more aggressively than regular TCP to achieve high network utilization. When the packet loss rate is higher than 10^{-3}, HighSpeed TCP's behavior is identical to regular TCP, improving compatibility with legacy TCP variants. HighSpeed TCP is now an Internet RFC, but it has not yet seen significant adoption.

Scalable TCP [24] builds upon the work of HighSpeed TCP. Its congestion window follows a Multiplicative Increase Multiplicative Decrease (MIMD) rule. Like HighSpeed TCP, Scalable TCP's window will grow at a faster rate after it has reached a certain threshold. Unlike HighSpeed TCP, however, ScalableTCP uses static increase and decrease parameters.

HighSpeed and Scalable TCP can both achieve good scalability and link utilization. However, they present some fairness concerns [36, 50], when flows with different RTTs compete for network bandwidth. Recall that standard TCP is biased towards flows with smaller RTTs. A disadvantage in both protocols is that their more aggressive nature makes them more prone to RTT unfairness [32].

TCP-Westwood+ is proposed in [9]. It uses the idea of bandwidth estimation from earlier proposals such as [7] and [25]. The key insight is to use the arrival rate of acknowledgements to infer the current network bandwidth. Unlike standard TCP, which halves the congestion window on packet loss, TCP-Westwood uses the bandwidth estimate to set the congestion window and slow-start threshold to more appropriate values. This makes TCP-Westwood perform significantly better than Reno in heterogeneous networks. More recently, the authors of TCP-Westwood proposed a new variant called TCP-Westwood with agile probing and persistent noncongestion detection (TCPW-A) [44]. The proposed changes make TCP-Westwood more suitable for links with dynamic, high bandwith-delay products. Moreover, its authors state that TCPW-A can achieve good efficiency, without compromising fairness and stability.

Another algorithm, called Binary Increase Congestion (BIC) is proposed in [50]. It uses a novel congestion window growth mechanism that views congestion control as a searching problem. The basic idea of the algorithm is as follows. A minimum window is defined as the congestion window where packet loss is not yet detected. A target window is defined as the midpoint between the minimum window (i.e., the current congestion window, provided no loss occurred yet) and a maximum window. Initially, the maximum window is a large pre-defined constant, or else it is the window at which packet loss was previously detected. If the current congestion window is far from the target window, BIC will increase the congestion window faster. This is called the additive increase phase. When the congestion window approaches the target window, BIC will enter the binary search phase, where the window has a logarithmic growth. If the congestion window reaches maximum window, BIC will enter the max probing phase, and the window will grow exponentially until the new maximum window is found. This behavior sets BIC apart from other scalable protocols, such as HighSpeedTCP and ScalableTCP, where the growth rate is always fastest near the maximum window. Its authors argue that, compared to TCP, BIC's design improves link efficiency, TCP friendliness, fairness, and protocol stability. BIC was made the default TCP congestion control algorithm in the 2.6.8 Linux kernel.

CUBIC is a less complex and more network friendly variant of BIC, proposed after some concerns were raised about the aggressiveness of BIC in networks with low round-trip times or low link speeds [18]. CUBIC eliminates BIC's three distinct window growth

phases. Instead, it uses a single cubic function that behaves in a similar way. Also, inspired on the ideas proposed in [29] and [19], CUBIC makes its congestion window growth rate independent of RTT, and instead dependent on the time since the last congestion event. CUBIC's authors state that its cubic window function ensures intra-protocol fairness, and that its RTT-independent congestion window increase function improves RTT fairness.

CUBIC distinguishes itself from many end-to-end congestion control algorithms proposed in the literature by the fact that it is the default congestion control algorithm in the Linux kernel, since version 2.6.19. As a consequence, it is one of the few proposals which has seen extensive real-world testing. Nevertheless, CUBIC it is not without its critics. Some researchers have pointed out that CUBIC can be overly aggressive to cross-traffic (e.g., VoIP calls) [49], have very slow convergence times, or not converge at all, showing significantly throughput unfairness in several network scenarios [30].

Satellite links

Several contributions have been proposed to address TCP's specific problems on satellite links. The proposal in [1] uses dummy segments to probe for available networks resources; it also introduces two new algorithms to optimize link utilization. In [8], along with several mechanisms to improve link efficiency, it is proposed that, to achieve better fairness, the window of flows with higher RTTs should increase faster than flows with smaller RTTs.

3.2 Delay-based congestion control algorithms

Delay-based algorithms mark a significant departure from the traditional loss-based approach. They are based around the premise that there are signs before network congestion occurs. So, they use network delay information, usually in the form of RTTs, to infer the state of queues at the routers. One advantage of the delay-based approach is that it is much more network friendly than the loss-based counterpart, in the sense that network congestion can be proactively avoided. Therefore, delay-based schemes can avoid the typical congestion window oscillation of loss-based algorithms, e.g., TCP NewReno's familiar "saw effect", and enable better link bandwidth utilization. However, as doubts still persist on the effectiveness of delay-based strategies, several authors argue that more research is needed [39].

Pure delay-based algorithms

There are several early proposals for congestion control algorithms based on delay information. In [46], RTT variations are used to control the congestion window size. The window grows according to Reno. But, if the value of the current measured RTT is higher than the average of the minimum and maximum measured RTTs so far, then the window size is decreased. In [45], variations in the sending rate are used to control the window size. The window is increased every RTT by one segment. The current throughput is then compared to the throughput achieved with the previous window. If the difference in the sending rate is less than a certain threshold, then the window is decreased by one segment. In [22], the window size is adjusted based on both RTT and window size.

Vegas [7], is a seminal work on end-to-end delay-based TCP congestion control that has directly influenced many delay-based proposals. Inspired on [45], Vegas measures the sending rate to control the congestion window size. Additionally, Vegas presents new ideas to improve packet loss recovery, and introduces changes to the slow-start mechanism to reduce packet loss. It can significantly reduce network congestion, and maintain the congestion window around the optimal value for a given link. Its linear growth, however, makes Vegas unsuitable for networks with high bandwith-delay products. Furthermore, Vegas' performance is hindered when it co-exists with more aggressive loss-based TCP variants, such as Reno.

FAST TCP, which can be considered an high-speed descendant of Vegas, is proposed in [47]. Even though FAST TCP builds upon the principles of Vegas, it increases the congestion window more aggressively to achieve good efficiency in high-speed networks. FAST TCP converges quickly to near the maximum window, and then, as the RTT increases, converges more slowly to the target window. Moreover, FAST TCP implements a rate pacing technique to solve the

burstiness effects observed with very large congestion windows. The authors claim that FAST TCP can achieve excellent link utilization in high-speed networks, converges to fairness quickly, and is stable. Nevertheless, it has been observed that FAST TCP lacks its fairness properties in some scenarios [36, 32]. FAST can also be problematic when there are many flows competing for bandwidth. Since each flow aims to queue α packets in the bottleneck router, the buffer may not be sufficient to hold all necessary $n \times \alpha$ packets, as n increases.

Hybrid-based algorithms

Hybrid-based TCP congestion control algorithms have become an active topic of research in the last decade. Their mixed approach promises increased link efficiency, while retaining some of the fairness and network friendliness properties of pure delay-based proposals. Moreover, their loss-based component allows hybrid-based algorithms to remain aggressive enough while competing for bandwidth with pure loss-based algorithms.

TCP-Illinois [33] follows an AIMD algorithm, but uses delay estimates to set the increase and decrease congestion window parameters. If TCP-Illinois does not detect queuing delay (i.e., network congestion), the increase parameter is set to the maximum value, making the congestion window grow quickly. As the queuing delay starts to build up, the increase parameter is then gradually decreased, making the congestion window grow more slowly. On packet loss, its decrease factor is adjusted according to the measured delay. An higher delay results in a larger decrease factor, and vice-versa. This improves its behavior in heterogeneous networks. Nevertheless, some concerns have been raised regarding TCP-Illinois' scalability in networks with very high bandwith-delay products due to its fixed maximum window increase parameter (10 packets per RTT), and in the presence of reverse path traffic [28].

Compound TCP is proposed in [43] to improve link efficiency and RTT fairness. It maintains two auxiliary congestion control windows. A traditional loss-based congestion window that follows Reno's AIMD behavior, and a scalable delay-based window inspired in Vegas. The resulting congestion window is the sum of the loss-based and the delay-based windows. If the network link is under-utilized, the delay-based component will quickly increase the congestion window to use available bandwidth, using a behavior similar to [13]. When congestion starts to build up, the delay-based component will reduce the window. Under heavy network congestion, Compound TCP reverts to Reno behavior. Compound TCP authors claim that the delay-based component allows it to achieve good link efficiency and RTT fairness. Moreover, since the throughput is lower bounded by the loss-based component, it solves Vegas' unfairness issues with loss-based schemes. It has been observed, however, that Compound TCP can revert to a Reno-like scaling behavior, even with light reverse traffic [28], and can suffer from fairness and scalability issues in links with very high bandwith-delay products [48]. Even though disabled by default, Compound TCP is present in Windows Server 2008, Windows Vista, and Windows 7.

The proposal in [23] is similar to Compound TCP in the sense that it also maintains a delay-based and a loss-based windows to achieve good link efficiency. The approach in [4] uses a Vegas-inspired, delay-based, MIMD mechanism while on high-speed mode. In [26], the algorithm switches between Reno and delay-based HighSpeed TCP according to the measured delay. The mechanism described in [42] aims to improve RTT fairness, and friendliness towards Reno, using delay measurements to control Reno's AIMD parameters. The authors of [34] propose an algorithm that combines the delay measures from [42] and the rate estimates from [9] to create an high-speed, RTT fair, and TCP friendly protocol. TCP-LP [27], a low priority TCP variant, introduces a novel one-way delay estimate technique. The proposal in [17], while not aiming for high-speed link efficiency, combines ideas from Reno and Vegas to improve Reno's behavior in heterogeneous networks.

4 Contributions of this work

In this work we explore current, and introduce new, delay-based techniques in order to improve an existing loss-based, end-to-end, congestion control algorithm. We believe that such techniques can bring significant performance advantages in numerous network scenarios. We run simulations with the Network Simulator 2 (NS-2) [37] tool to verify the performance improvements of our proposals. The simulation results focus on key performance metrics such as link efficiency, fairness, router queue occupancy, latency, and packet loss rate. Finally, we compare our proposals with several other high-speed congestion control algorithms.

In order to evaluate our proposals, we extend and improve the Hamilton Institute benchmark TCP suite [5] to fulfill our requirements. The test suite has three different purposes. First, to assess the performance of existing TCP congestion control algorithms and establish a baseline. Next, to test the new candidate algorithms and modifications under identical test scenarios. Finally, to compare the obtained results for the new algorithms with the current proposals.

The tests focus on three different vectors. First, the efficiency of the algorithms is assessed. Efficiency is concerned with link utilization, therefore a high-speed TCP variant must be able to use the available capacity of networks with high bandwith-delay products. Second, fairness is measured. We expect that flows competing for bandwidth in a bottleneck link eventually converge to a fair share of the bandwidth. Flow fairness is assessed under different scenarios, such as flows with different RTTs. Fairness is measured according to Jain's fairness index [10]. Finally, network friendliness is studied by inspecting packet loss rates, queueing delay, and router queue occupancy. Network friendliness is a desirable characteristic of a congestion control algorithm, as it reduces network strain, and improves the performance of delay-sensitive applications, such as VoIP and interactive sessions.

We chose the Network Simulator Version 2 (NS-2) [37] tool to implement our proposals and perform the benchmarks. NS-2 is a discrete event network simulator that is widely used in the network research community. It is considered by many a non-commercial "standard" for the evaluation of networks and networking algorithms. NS-2 supports a wide range of simulation scenarios, including wired and wireless networks, and protocols such as TCP. NS-2's core is written in C++, and has an OTcl interface to specify various network scenarios – ranging from simple dumbbell to more complex topologies. Because of its open nature and very flexible architecture, NS-2 has been constantly developed and improved. Therefore, choosing NS-2 for the implementation and evaluation of a new proposal gives other researches the opportunity to validate the results, and improve the acceptance of such proposal.

We obtained through simulation encouraging results, which means that, at least for the simulated network environments, it is possible to have reasonable end-to-end congestion control algorithms. At the same time, with these mechanisms, the complexity is transferred to the edge of the network and compatibility with existing TCP versions is maintained.

References

[1] I. Akyildiz, G. Morabito, and S. Palazzo. TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks. *IEEE/ACM Transactions on Networking*, 9(3):307–321, June 2001.

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

[3] A. Almeida and C. Belo. Analysis and Implementation of a Stateless Congestion Control Algorithm. In *SPECTS'05*, July 2005.

[4] A. Baiocchi, A. Castellani, and F. Vacirca. YeAH-TCP: Yet Another Highspeed TCP. In *proc. International Workshop on Protocols for Fast Long-Distance Networks*, Marina del Rey, California, USA, February 2007.

[5] Hamilton Institute TCP benchmark suite. `http://www.hamilton.ie/net/tcptesting.zip`.

[6] B. Braden, D. Clark, and J. Crowcroft. Recommendations on Queue Management and Congestion Avoidance in the Internet. IETF RFC 2309, April 1998.

[7] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[8] C. Caini and R. Firrincieli. TCP Hybla: a TCP enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22(5):547–566, September 2004.

[9] C. Casetti, M. Gerla, S. Mascolo, M. Sansadidi, and R. Wang. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks Journal*, 8:467–479, 2002.

[10] D. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17(1):1–14, June 1989.

[11] J. Kenney (editor). *Traffic Management Specification, Version 4.1.* ATM Forum Technical Committee, AF-TM-121.000, March 1999.

[12] A. Falk and D. Katabi. Specification for the Explicit Control Protocol (XCP). IETF Internet Draft, November 2006.

[13] S. Floyd. HighSpeed TCP for Large Congestion Windows. IETF RFC 3649, December 2003.

[14] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7:458–472, 1999.

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

[16] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[17] C.P. Fu and S.C. Liew. TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks. *Selected Areas in Communications, IEEE Journal on*, 21(2):216–228, Feb 2003.

[18] S. Ha, I. Rhee, and L. Xu. CUBIC: a New TCP-Friendly High-Speed TCP Variant. *SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[19] T. Hatano, M. Fukuhara, H. Shigeno, and K. Okada. TCP Congestion Control Realizing Fairness with TCP Reno in High Speed Networks. *IEIC Technical Report (Institute of Electronics, Information and Communication Engineers)*, 103(198):19–24.

[20] International Telecommunication Union (ITU-T). Recommendation G984.1 "Gigabit-capable Passive Optical Networks (GPON): General characteristics", March 2003.

[21] V. Jacobson. Congestion Avoidance and Control. In *proc. ACM SIGCOMM*, pages 314–329, Stanford, CA, August 1988.

[22] R. Jain. A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks. *ACM Computer Communication Review*, 19(5):56–71, October 1989.

[23] K. Kaneko, T. Fujikawa, Z. Su, and J. Katto. TCP-Fusion: A Hybrid Congestion Control Algorithm for High-Speed Networks. In *proc. International Workshop on Protocols for Fast Long-Distance Networks*, Marina del Rey, California, USA, February 2007.

[24] T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *Computer Communication Review*, 33(2):83–91, April 2003.

[25] S. Keshav. Packet-Pair Flow Control. *IEEE/ACM Transactions on Networking*, 1995.

[26] R. King, R. Baraniuk, and R. Riedi. TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP. In *proc. IEEE INFOCOM*, Miami, FL, March 2005.

[27] A. Kuzmanovic and E.W. Knightly. TCP-LP: Low-Priority Service Via End-Point Congestion Control. *Networking, IEEE/ACM Transactions on*, 14(4):739–752, August 2006.

[28] D. Leith, L. Andrew, T. Quetchenbach, and R. Shorten. Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms. In *proc. International Workshop on Protocols for Fast Long-Distance Networks*, Manchester, UK, March 2008.

[29] D. Leith and R. Shorten. H-TCP Protocol for High-Speed Long Distance Networks. In *proc. International Workshop on Protocols for Fast Long-Distance Networks*, Argonne, Illinois, USA, February 2004.

[30] D. Leith, R. Shorten, and G. McCullagh. Experimental Evaluation of Cubic TCP. In *proc. International Workshop on Protocols for Fast Long-Distance Networks*, Los Angeles, USA, 2007.

[31] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[32] Y.T. Li, D. Leith, and N.R. Shorten. Experimental Evaluation of TCP Protocols for High-Speed Networks. *IEEE/ACM Transactions on Networking*, 15(5):1109–1122, 2007.

[33] S. Liu, T. Başar, and R. Srikant. TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks. In *proc. International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, Pisa, Italy, October 2006.

[34] C. Marcondes, M.Y. Sanadidi M., Gerla, and H. Shimonishi. TCP Adaptive Westwood – Combining TCP Westwood and Adaptive Reno: A Safe Congestion Control Proposal. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 5569–5575, May 2008.

[35] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. IETF RFC 2582, October 1996.

[36] S. Molnár, B. Sonkoly, and T. A. Trinh. A Comprehensive TCP Fairness Analysis in High Speed Networks. *Computer Communications*, 32(13-14):1460–1484, August 2009.

[37] The Network Simulator 2 (NS-2) Web page. `http://www.isi.edu/nsnam/ns/`.

[38] J. Postel. Transmission Control Protocol. IETF RFC 793, September 1981.

[39] R.S. Prasad, M. Jain, and C. Dovrolis. On the Effectiveness of Delay-Based Congestion Avoidance. In *Second International Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[40] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. IETF RFC 3168, September 2001.

[41] D. Reed, J. Saltzer, and D. Clark. Active networking and end-to-end arguments, 1998.

[42] H. Shimonishi, T. Hama, and T. Murase. TCP-Adaptive Reno for Improving Efficiency-Friendliness Tradeoffs of TCP Congestion Control Algorithm. In *proc. International Workshop on Protocols for Fast Long-Distance Networks*, Nara, Japan, February 2006.

[43] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *proc. IEEE INFOCOM*, Barcelona, Spain, April 2006.

[44] R. Wang, K. Yamada, M. Sanadidi, and M. Gerla. TCP With Sender-Side Intelligence to Handle Dynamic, Large, Leaky Pipes. *IEEE Journal on Selected Areas in Communications*, 23(2):235–248, February 2005.

[45] Z. Wang and J. Crowcroft. A New Congestion Control Scheme: Slow Start and Search (Tri-S). *ACM Computer Communication Review*, 21(1):32–43, January 1991.

[46] Z. Wang and J. Crowcroft. Eliminating Periodic Packet Losses in 4.3-Tahoe BSD TCP Congestion Control Algorithm. *ACM Computer Communication Review*, 22(2):9–16, April 1992.

[47] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Transactions on Networking*, 14(6):1246–1259, 2006.

[48] X. Wu. A Simulation Study of Compound TCP. Technical report, National University of Singapore, July 2008.

[49] X. Wu. Safely Speeding Up Bandwidth-greedy and Elastic Applications of the Internet. Technical report, National University of Singapore, April 2009.

[50] L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524 vol.4, March 2004.