A Survey of Delay-Based and Hybrid TCP Congestion Control Algorithms

Rasool Al-Saadi, Grenville Armitage, Jason But and Philip Branch

***Abstract*—Congestion Control (CC) has a significant influence on the performance of Transmission Control Protocol (TCP) connections. Over the last three decades, many researchers have extensively studied and proposed a multitude of enhancements to standard TCP CC. However, this topic still inspires both academic and industrial research communities due to the change in Internet application requirements and the evolution of Internet technologies. The standard TCP CC infers network congestion based on packet loss events which leads to long queuing delay when bottleneck buffer size is large. A promising solution to this problem is to use the delay signal (RTT or one-way delay measurements) to infer congestion earlier and react to the congestion before the queuing delay reaches a high value. In this survey paper, we describe the delay signal and the algorithms that completely or partially utilise this type of signal. Additionally, we illustrate standard CC and modern Active Queue Management (AQM) principles and discus the interaction between AQM and the delay signal.**

***Index Terms*—TCP, Delay-based congestion control, Active Queue Management**

I. Introduction

Transmission Control Protocol (TCP) [1] deserves a great tribute for the success of the internet in the last three decades because of its ability to generally perform sufficiently well in spite of the significant changes in internet technologies. TCP is a transport protocol that provides byte-streams and reliable data transfer over the packet-based best-effort Internet Protocol (IP) layer [2]. As the preferred transport for many internet applications, TCP has earned a lot of attention from the research community keen to maximise the protocol's performance. Congestion Control (CC) is a critical part of TCP that directly influences the protocol's performance. CC aims to manage network resources in an efficient manner and to provide resource sharing among competing flows while protecting the network from collapse.

Typically, TCP CC probes a path's capacity by sending data and monitoring the incoming implicit (or sometimes explicit) feedback signal. Based on the feedback signal, TCP reduces or raises the number of unacknowledged bytes in flight to minimise congestion while achieving high link utilisation. Implicit feedback can be inferred from packet loss caused by bottleneck's buffer overflow (loss-based CC) or variations in packet delivery delay caused by bottleneck queue building up (delay-based CC). Additionally, TCP CC can benefit from explicit feedback, such as Explicit Congestion Notification (ECN) [3], where end hosts and bottlenecks both support such a feature.

The most common, actively used TCP implementations utilise reliable and easy to implement loss-based CC algorithms. However, flows using loss-based CC become problematic when they compete with latency-sensitive flows for capacity at bottlenecks having large buffers. In such case, loss-based TCP pushes the bottleneck queue to a high threshold until buffer overflow occurs causing long queuing delay. The resulting additional latency can negatively impact latency-sensitive applications (such as live video streaming and online gaming) and degrade network quality of service in general. Using packet loss as a signal also wastes network resources (due to the need for retransmitting the lost packet).

Consequently, many researchers have explored the use of delay as a feedback signal (partially or fully) to remedy the shortcomings of loss as a signal. Delay-based CC approaches provide low latency data transfer through controlling network congestion as soon as bottleneck queues start building up. Using a such strategy, they prevent packet losses which can significantly affect unreliable UDP streams such as VoIP traffic. Additionally, many delay-based CC algorithms aim to achieve high and stable throughput by reducing oscillation in data sending. Oscillation in packet sending reduces the performance of transport protocol in long-distance high-speed paths with shallow bottlenecks buffers. Delay-based CC approaches can be optimised to be used efficiently in high-speed long-distance networks to converge to full link utilisation quickly without stressing the network. Moreover, it has been shown in many industrial and academic works that delay-based CC can be efficiently used in background bulk data transfer transports and scavenger class services such as system updates [4], [5], [6], [7], [8], [9]. It is also possible to utilise the delay signal for distinguishing between congestion related and random losses [10], [11]. This is useful to achieve high link utilisation in lossy networks.

Unfortunately, delay-based feedback is complicated and fraught with difficulties including noise in the signal, sampling problems, coexistence and many other issues that this paper covers. This prevents delay-based CCs from being widely used for general purpose control congestion.

Developing good CC strategies is a complicated task, as it

R. Al-Saadi is with the College of Science, Al-Nahrain University, Baghdad, Iraq (e-mail: rha@sc.nahrainuniv.edu.iq), and also with with the Internet For Things (I4T) Research Group, School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia (e-mail: ralsaadi@swin.edu.au).
G. Armitage is Engineering Manager of the Transport Protocol R&D team at Netflix Inc, and Adjunct Professor with the School of Software and Electrical Engineering, Swinburne University of Technology, Australia (e-mail: garmitage@swin.edu.au).
J. But and P. Branch are with the Internet For Things (I4T) Research Group, School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia (jbut@swin.edu.au, pbranch@swin.edu.au).

requires intelligent awareness of network resources availability and using these resources in an efficient and fair manner. At the same time finding an effective solution is highly desired for many applications and users. Many improvements to standard TCP CC have been suggested, but no technique is perfect yet in all situations.

Much work has been done to study different congestion control schemes. Hasegawa and Murata [12] study the fairness issues in TCP CC and the available solution that can improve the fairness. Afanasyev et al. [13] study different host-to-host congestion control techniques classified based on the goals such as improving protocol performance in networks with high packet reordering, wireless networks or high-speed and long-distance networks. Ros and Welzl [14] focus on low priority end-to-end CC techniques used for background data transfer. Lochert et al. [15] review congestion control techniques for mobile ad-hoc networks. Callegari1 et al. [16] study thirteen different TCP CC variants implemented in the Linux Kernel 2.6.x.

Unlike previous work, we present a survey of congestion control techniques that utilise delay signal as a primary or secondary indicator to control network congestion. We describe general principles of TCP CC and congestion signal types, and explore the challenges of using delay signal and how some recently popularised queuing-delay based Active Queue Management (AQM) techniques are likely to interact with delay-based CC techniques. Since there are many proposed TCP CC utilising the delay signal, this paper covers popular techniques that have real impact on their working environments.

The rest of this paper is structured as follow. Section II provides principles of TCP flow control and congestion control. Section III describe the interaction between TCP and the bottleneck FIFO buffer and introduces AQM functionality. Section IV is devoted to TCP congestion control literature including the delay and loss congestion feedback signals and standard TCP CC algorithms. Section V is dedicated to reviewing popular delay-based, hybrid and delay-sensitive TCP variants. Section VI discusses the challenges faced with using the delay signal including deploying AQM. We conclude the paper with Section VII.

II. Transmission Control Protocol – An Overview

The TCP layer provides a reliable, connection-oriented end-to-end transport protocol that guarantees error-free, in order delivery of data to the destination [1]. TCP flow control and congestion control limit the amount of outstanding (unacknowledged) sent data. Flow control prevents fast senders from overrunning the buffers of slow receivers (which causes packet loss). Congestion control aims to prevent senders from sending too much data that can overflow buffers within the network (network congestion). In this section, we summarise the principles of TCP flow control, TCP congestion control and how the injection of packets into the network can be controlled.

A. TCP Reliability and Flow Control

The IP layer [2] provides a best-effort packet transfer service between the source to destination host. IP does not guarantee delivery, nor ensure packets are delivered in-order. TCP is responsible for both data integrity and network resource management to provide a reliable end-to-end connection.

Data transfer over TCP is initiated by an application which supplies data to the TCP stack. TCP buffers the data, allocating each byte a sequence number. TCP then partitions the buffered data into segments, assigning each segment the sequence number of the first byte in that segment. Using a sliding window mechanism (shown in Figure 1), TCP transmits a number of segments to the receiver over the IP protocol.

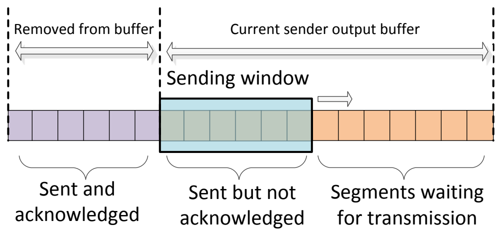

Removed from buffer
Current sender output buffer
Sending window
Sent and acknowledged
Sent but not acknowledged
Segments waiting for transmission

Fig. 1. TCP sliding window mechanism

At the receiver, the destination host buffers the segments in its TCP receive buffer. If a segment arrives without error and in order (checked using the sequence number), the receiver confirms receiving the segment by generating and sending back an acknowledgement packet (ACK)[1] containing the sequence number of next byte it is expecting to receive.

The ACK packet confirms the delivery of all bytes that have sequence numbers smaller than the ACK number. As such, it is not required to send an ACK packet for each data packet. When an ACK packet is received by the source, the sender moves the sending window (*swnd*) by the amount of acknowledged data and sends new segments if they are ready in the sending buffer. In other words, the sender should not transmit more data than *swnd* allows until receiving acknowledgement of previously sent data i.e. $seq_{next} < seq_{una} + swnd$ where seq_{next} is the sequence number of next packet to be sent, and seq_{una} is the sequence number of the first packet sent but not yet acknowledged. As such, *swnd* effectively limits the number of unacknowledged bytes (bytes in-flight).

Using this mechanism, there is a period of time before the sender is aware if a segment arrived at the destination correctly. The earliest the sender can receive an ACK is given by the round trip time (RTT) – the combination of serialisation delays (transmission time), propagation delays, and bottleneck queuing delays along both the forward and reverse paths. In reality each ACK may be further delayed by additional factors such as the processing power of the end hosts and middleboxes along the path, and efficiency improvement mechanises (e.g. delayed ACK).

We can conceptualise the link between the source and destination as a virtual pipe between the two points (see Figure 2). The bandwidth (link capacity) is represented by the diameter of the pipe, while the delay (path RTT) is

[1]A normal TCP packet with the ACK flag in TCP header set.

represented by the length. The product of these two values, the bandwidth-delay product (BDP), represents the pipe's volume – the amount of data that saturates the link between the sender and receiver.

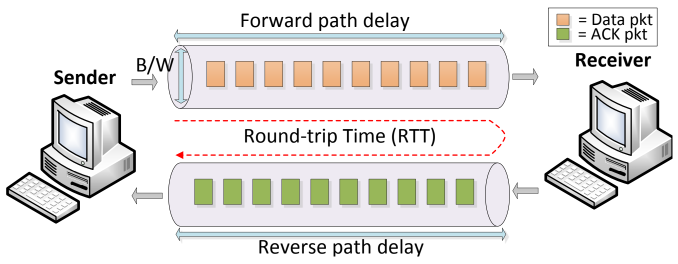


Fig. 2. Bandwidth-Delay Product and link pipe

If the sender fills the pipe completely, data transmission will be continuous because acknowledgements will be received while data is still being sent. On the other hand, if the pipe is partially filled, there will be stalls during transmission because the sender has to wait for acknowledgements to trigger sending new data as shown in Figure 3. In order to achieve optimum throughput, the sender should keep *swnd* greater than or equal to the link BDP.

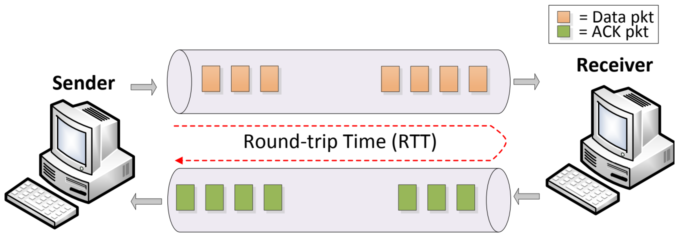


Fig. 3. Unfilled link pipe causing low link utilisation

The destination TCP receive buffer size is subject to memory availability, system/application configuration and processing power. Therefore, the sender should be aware of the receiver buffer availability. As part of TCP flow control, the receiver specifies the maximum number of bytes it is willing to receive via the advertised receive window size (*rwnd*). This mechanism prevents a fast sender from exhausting the limited buffer size in a slow receiver.

B. TCP Congestion Control

The original TCP specification limits outstanding (unacknowledged) packets only by the receiver's *rwnd* [1]. However, this completely ignores network congestion.

Without awareness of the current network congestion state, TCP may send more data than a bottleneck can handle, resulting in heavy packet loss, significant reduction in network performance, increase in packet delivery delay, and could lead to a phenomenon called *congestion collapse*. This is a scenario where only a small portion of transmitted data is successfully delivered and acknowledged, leading to low goodput[2] and long queuing delays.

Congestion collapse in the Internet was first observed in the mid 1980s [17], due to TCP senders spuriously retransmitting packets that were actually not missing but waiting in long queues. The retransmissions exhausted bottleneck capacity more seriously as the number of flows increases.

An early solution to mitigate the congestion problem relied on an explicit message sent using Internet Control Message Protocol (ICMP) [18]. An ICMP Source Quench [1] message would be sent by the congested router to the sender when the bottleneck buffer becoming congested, causing the sender to throttle back. However, use of ICMP Source Quench was deprecated due to ineffectiveness and unfairness issues [19].

The fundamental solution has been the development and use of end-to-end congestion control (CC). CC algorithms aim to monitor the network's current congestion state, and to use this information to adjust the sending rate, directly or indirectly, to stabilise network usage, maintain high link utilisation and provide a fair share of the available bandwidth [20].

TCP CC [21] maintains a congestion window size (*cwnd*) for each TCP flow, representing the maximum number of bytes the sender may send and have outstanding (unacknowledged) based on current network congestion state. The sender selects the minimum of *rwnd* and *cwnd* as the final sending window size $swnd = min(rwnd, cwnd)$.

TCP CC attempts to fully utilise the network without causing congestion by inferring current network conditions and dynamically adjusting *cwnd* accordingly. It reacts to changing network conditions by increasing *cwnd* when no congestion is detected and reducing it when a congestion event occurs.

At the start of new TCP connection, the sender is unaware of available network bandwidth. TCP uses a phase called slow-start (see Section IV-C1) to quickly probe the available bandwidth in the path. During slow-start, the sender increases *cwnd* on each ACK received. When congestion is detected, TCP CC sets *cwnd* as a portion of the achieved window size when the congestion was detected, and exits slow-start to enter another phase called congestion avoidance.

During congestion avoidance, *cwnd* increases more slowly than in slow-start, typically by one segment per RTT, to avoid causing congestion while still adapting the window size to any changes in available capacity. A detailed discussion on TCP CC algorithms is available in Section IV-C.

Controlling congestion efficiently is not an easy task due to the distributed nature of the TCP/IP protocol and constantly changing network conditions. Not all CC algorithms have the same goals nor are expected to function in all environments. Some algorithms may prioritise minimising delays, while others may focus on performing well under special conditions such as within a data centre. Nevertheless, listed below are some common goals shared by most CC algorithms [17].

- Preventing congestion collapse: Considered the main reason for the existence of CC.

[2]Goodput is the amount of data that arrives to the destination successfully

- High bandwidth utilisation: Considered a fundamental requirement for all CC algorithms. CC should avoid path capacity underutilisation to maximise throughput.
- Fairness: CC should guarantee an acceptable equal share of the available bandwidth among all competing flows sharing the same bottleneck.
- Fast convergence to fairness: When a new TCP flow joins a shared bottleneck, the CC should react rapidly to this event by increasing the *cwnd* of the new flow and reducing it for all other flows until fairness is achieved.
- TCP-Friendly: For deployment purposes, CC algorithms intended to be used in an uncontrolled network, e.g. the Internet, should coexist with other CC algorithms by maintaining fairness.

Due to differences in CC algorithms, factors such as path RTT and flow starting sequence may affect fairness, giving advantages to some flows over others. For example, TCP Reno [21] flows transmitted over a high RTT path get lower overall throughput than flows using shorter RTT paths. This is due to the Reno algorithm increasing *cwnd* by one Maximum Segment Size (MSS) every RTT. As a result, *cwnd* increases faster for flows using shorter paths.

TCP CC protocol variations should manage *cwnd* such that it does not reduce their – or other standard CC protocols – performance significantly. A well-known example of incompatibility with standard CC is TCP Vegas [22] which realises low throughput when competing with loss-based CC in a large bottleneck buffer environment [23].

Some CC algorithms have been designed to work in a controlled environment, such as data centres, where all machines use the same algorithm. For these protocols, compatibility with other widely used algorithms is not a concern.

There are arguments about the ideology of flow fairness in TCP CC and how the fairness concerns benefits for users but not for individual flows in real world [24]. However, flow fairness is still considered important for many CC algorithms studied in academia. As such, CC algorithms typically take flow fairness into consideration, trying to distribute available bandwidth amongst competitive flows fairly.

Alternatively, Low Priority Congestion Control (LPCC) algorithms specifically aim to achieve lower capacity sharing of the available bandwidth and/or lower queuing delay when coexisting with other flows [14]. This group of algorithms (also called *scavenger class* or *less than best effort* service) are used by background applications such as bulk file transfer, peer-to-peer applications and automatic software updates. In these cases, the algorithms try to reduce the impact on higher priority foreground applications, while attempting to maintain fairness when coexisting with flows from the same class.

The original TCP specifications [1] do not specify a direct mechanism for the hosts to learn about network congestion state. As a result, TCP CC utilises indirect information to determine whether the path is congested or not, or the degree of congestion in the path. This indirect information gathered from measurements taken during packet exchange between hosts, and typically varies due to buffering effects that may occur at any point in the network path.

During the last three decades, many congestion control algorithms have been proposed to calculate an optimum *cwnd*. However, only a few have been standardised including TCP Reno [21], TCP NewReno [25] and TCP SACK[26]. We denote these CC algorithms as standard TCP in this paper.

C. Controlling the injection of packets into the network

Most TCP implementations utilise window-based CC strategies to limit the number of injected packets in the network. Although this mechanism is efficient, easy to implement and does not require timers, it can generate periodic packet bursts into the network. This can lead to delay fluctuations, increased packet losses, higher queuing delays, and lower throughput [27]. These bursts occur because the transmitter immediately sends new packets (as many as *swnd* allows) whenever an acknowledgement is received. If acknowledgements are delayed or compressed for any reason (e.g. congestion in the reverse path), the sender will receive multiple acknowledgements in a short period, freeing a space in the window and causing the sender to transmit multiple packets in a burst.

An alternative approach to control the number of transmitted packets in the network is to limit the actual sending rate directly. Rate-based CC can calculate the required sending rate that would fully utilise available bandwidth without causing congestion. The calculated sending rate is then used to schedule regular transmission of packets, removing the burstiness seen with the sliding window. Other types of rate-based CC can estimate the required sending rate in a similar way to window-based CC, i.e. increase the sending rate when no congestion is detected and reduce it when congestion happens to achieve acceptable fairness when competing with window-based flows.

In rate-based strategy, the required sending rate can be realised either inside TCP stack or externally (e.g. packet schedulers) using packet pacing. Packet pacing allows a chunk of packets to be spread across a time slot by adding gaps between the sending of packets. The duration of these gaps is determined by the required sending rate.

Packet pacing is also used with window-based mechanisms to eliminate packet bursts [28]. In this case, the transmission of a window's worth of packet is spread across a full RTT.

Packet pacing provides smoother traffic flows and more stable demands on the network and can improve the stability of TCP by minimising variations in queue utilisation. Additionally, it has been shown that packet pacing provides a positive impact on delay-based CC by providing smooth RTT measurements [29].

Despite their benefits relative to window-based strategies, implementation of rate-based strategies is often more complex and requires accurate timers which is considered a costly requirement for embedded and low-end devices.

III. Buffering and Queue Management

Network buffers are used to absorb packet bursts, reduce packet losses, and improve overall network performance. They exist in many places of the packet transmission path including the host application, TCP socket, host network layer, network interface cards (NIC), network switches, routers, proxies and firewalls.

Buffers are used to temporarily queue packets when the next layer is busy or unable to process the packet as fast as they are provided. There may be a number of causes, such as devices with low processing power, network scheduling priority, temporary reductions in link layer sending rate, and transient network congestion.

A. Traditional Buffering and Queues

The most common method for implementing network buffers is First-In First-Out (FIFO) with a DropTail management mechanism. In a FIFO queue, packets are appended to the tail of the queue during the enqueue process and fetched and removed from the head of the queue during the dequeue process. When the queue size exceeds the buffer size, the DropTail mechanism drops any new packet until suitable buffer space becomes available. Figure 4 shows a conceptual representation of FIFO and DropTail mechanism.

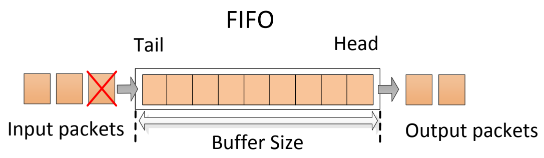


Fig. 4. FIFO and DropTail buffer management

When TCP was first designed, the bit error rate of transmission channel (usually wired) was very low. Therefore, packet loss was mainly caused by buffer overflow, and taken as an indication of congestion at the bottleneck. This relationship between packet loss and network congestion is exploited by loss-based TCP CC to infer congestion along the path.

The proliferation of oversized FIFO buffers in the network, coupled with the aggressiveness of loss-based TCP CC, causes high queuing delay in a phenomenon called Bufferbloat [30]. This high delay has a negative impact on latency-sensitive applications in particular, and on network performance in general.

Active Queue Management (AQM) is a mechanism used to keep the bottleneck queues of network nodes to a controlled depth, effectively creating short queues [31]. AQM is used as a replacement for the DropTail mechanism. When AQM detects congestion it reacts by dropping or marking packets with an ECN [3]. The loss event or ECN signal is then detected by the sender which reduces the transmission rate by decreasing *cwnd*.

B. Active Queue Management

In the last two decades, many AQM algorithms have been proposed to manage the queuing delay problem. However, none have yet been widely deployed due to both a reduction in network utilisation and complicated optimal configuration.

Legacy AQMs monitor queue occupancy based on bytes or packets in the queue. If the queue length becomes larger than a specific threshold, AQM infers congestion and reacts accordingly based on the congestion level. A well-known example of such statistical AQM is Random Early Detection (RED) [32]. Many queue occupancy-based AQMs have been proposed to mitigate different issues [33], [34], [35].

More recently, new AQM mechanisms have emerged that rely on queue *delay* measurement rather than queue occupancy. Queue delay is directly correlated to the network metric that AQM is intended to manage. These new AQMs are able to achieve high throughput and better delay control with low complexity. Further, these AQMs are designed to perform reasonably using their default configurations. Well-known examples of such AQMs are CoDel (Controlled Delay) [36] and PIE (Proportional Integral controller Enhanced) AQM [37], [38].

Moreover, hybrid AQM/scheduler schemes have been proposed to improve fairness between competing flows while keeping queuing delay low. They achieve these goals by diverting the flows into separately managed queues and applying an individual AQM instance for each queue. This separation protects low rate flows from aggressive flows while the individual AQM instances control the queue delay. Examples of hybrid AQM/scheduler schemes are FQ-CoDel (Flow-Queue CoDel) [39] and FreeBSD's FQ-PIE (Flow-Queue PIE) [40]. In addition to control queuing delays and provide relatively equal sharing of the bottleneck capacity, these AQMs provides short periods of priority to lightweight flows to increase network responsiveness.

Figure 5 illustrates simplified FQ-CoDel and FQ-PIE algorithms where flows are hashed to separate queues which are managed by either CoDel or PIE AQM. These queues are serviced using a deficit round robin scheduler with higher priority for new flows.

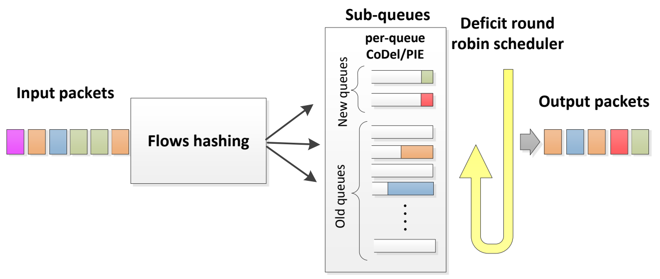


Fig. 5. Simplified FQ-CoDel/FQ-PIE AQMs

IV. TCP Congestion Control Literature -- Signals and Algorithms

In Section II we explained that CC algorithms try to estimate available bandwidth to optimally configure *cwnd* and maximise network utilisation. However, accurate bandwidth estimation is hard to achieve. Instead, CC algorithms use one or more congestion feedback signals to infer whether the path is under or over utilised. Senders react by increasing or reducing *cwnd* appropriately.

A. Implicit Congestion Feedback Signals

In many cases network infrastructure does not provide enough information to end hosts regarding the current network condition. As such, end hosts need to infer network state indirectly. One such mechanism is the congestion state of the path. Congestion state may be a simple binary signal or more advanced signal indicating the level of congestion. These signals are used to infer congestion without requiring support from middleboxes in the path between the sender and receiver. Typically, TCP uses either loss or delay signals to infer congestion.

1) The loss signal : Packet loss is used as a *loss signal* by many TCP CC algorithms to indicate network congestion. When a bottleneck experiences transient congestion, packets are queued until buffer space is exhausted. When this occurs, any new packets arriving at the bottleneck will be discarded until the queue drains, freeing up buffer space.

A sender typically detects packet loss using either the TCP Retransmission Time-Out (RTO) timer, or via three duplicate acknowledgement (3DUPACK) packets.

The RTO timer fires for a packet when no ACK is received for that packet, or any packet after it, within the RTO period of time. This event may occur during heavy loss situations due to severe network congestion and/or high noise in the link. RTO also commonly triggered due to loss of last packet in a window (tail drop) when the sender has nothing more to send for longer than an RTO period (application limited flows). For example, if the sender transmits three segments and the third segment lost, then there is no direct way for the receiver to inform the sender (using ACK) that it has not received last segment. Therefore, the sender will keep waiting unit RTO trigged.

Figure 6 illustrates a simple case of the RTO mechanism. No ACKs are received and the RTO timer causes the first packet to be retransmitted. The RTO timer must be tuned correctly to maximise link utilisation. Too large a value results in longer periods with no traffic, while too low a value wastes bandwidth due to multiple transmissions of successfully received packets. TCP Tahoe introduced improved RTO time estimation [41]. The original TCP CC specification included just the RTO mechanism to detect losses.

Alternatively, lost packets result in unordered packet arrival at the receiver. In this case, an ACK is constructed containing the sequence number of the missing packet for each subsequent packet arrival. With 3DUPACK, the sender uses the arrival of the third duplicate ACK packet (four identical ACKs) to infer that the corresponding packet was lost and trigger a retransmission [21].

Detecting congestion using 3DUPACK will typically occur more rapidly than waiting for the RTO timer to fire. This allows for a quicker recovery through the fast retransmission process as shown in Figure 7.

The 3DUPACK mechanism was introduced by TCP Tahoe [41]. TCP uses a threshold of three duplicate ACKs as a balance between the speed of loss detection and false positive detection due to the possibility of out-of-order delivery by the IP layer.

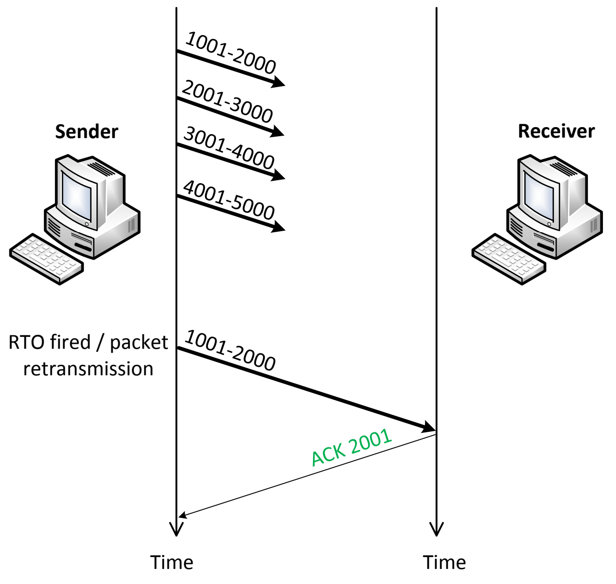


Fig. 6. A simplified Seq./ACK numbers timeline showing a case of TCP retransmission timeout

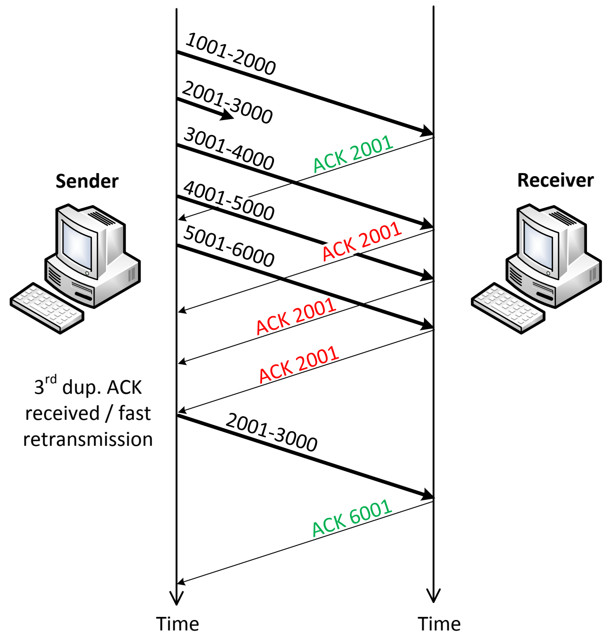


Fig. 7. A simplified Seq./ACK numbers timeline showing 3DUPACK and fast retransmission mechanisms

Unlike other inferred congestion signals, determining the *loss signal* does not require precise timers or time calculations, and is simple to implement with minimal code. This was important when TCP/IP was first developed as processing resources were more limited. The simplicity of the *loss signal* is why standard TCP (and key variants such as TCP CUBIC [42]) use it to infer congestion.

While using *loss* feedback is effective and easy to implement, it has some drawbacks. Firstly, using the *loss signal*, we can infer congestion only after network congestion becomes high enough to cause packet loss. The higher use of buffering capacity leads to longer queuing delays. This can lead to a poor quality of experience for latency sensitive applications sharing the bottleneck's buffer, such as VoIP and online gaming.

Secondly, packet loss is not always caused by network congestion. In some wireless networks packet loss may be caused by high bit error rate (BER) or user mobility [43]. The data link layer of modern wireless networks provides better reliability such that upper layers will rarely see packet losses on these networks. Link layer reliability instead translates into additional latency fluctuations at the IP layer, with subsequent implications for delay-based CC. These will be further discussed in Section VI-B. As such, while the performance of conventional TCP on older wireless networks can be significantly affected by random packet losses, this is not true for modern wireless networks.

2) The delay signal : Measurement of delay can also be used to infer network congestion. The delay can be directly measured in the form of RTT or One Way Delay (OWD), or calculated as a delay gradient signal to more directly measure changes to delay. Figure 8 illustrates the delay signal between two hosts connected over a bottleneck.

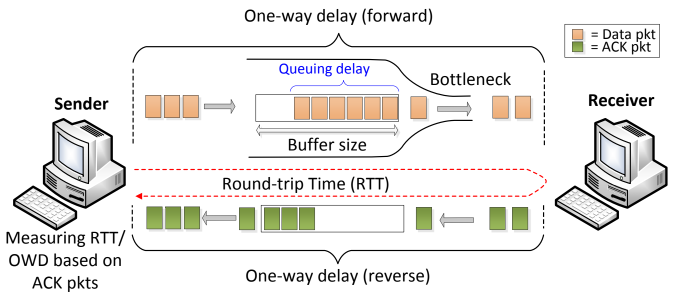


Fig. 8. Delay congestion feedback signal

Some studies argue that there is low correlation between measured delay and congestion in many cases [44], [45], [46]. However, McCullagh and Leith [47] studied the correlation between the delay signal and congestion and concluded that the flow's aggregate behaviour is what is important for delay-based CC, not a single observation by one flow. Also, Prasad et al. [48] investigated the reasons for the weak correlation between delays and losses caused by congestion and identify conditions under which the delay signals can fail to provide durable congestion feedback.

The *delay signal* provides more timely feedback of network congestion than the *loss signal* or even an explicit feedback signal. This is important for large BDP networks (also called Long Fat Networks LFNs) [49]. Unlike the *loss signal*, the *delay signal* gives approximate information on the degree of congestion such that the CC algorithm can proactively reduce the sending rate before packet loss occurs, or even before the queuing delay becomes high.

Moreover, the *delay signal* is also effective in lossy network environments. When packet loss occurs unrelated to congestion, packet delivery time will not increase and the *delay signal* is unaffected. As a result, a delay-based CC has the potential to be more tolerant to random packet losses and may perform better in lossy network environments.

Many delay-based CC algorithms use measured RTT as a delay signal, because this only requires TCP sender side modification. As a path's RTT combines the forward and reverse OWD, there is no way to distinguish what component of delay originates in the forward or reverse path. Including the reverse path delay in estimating queuing delay can lead to unnecessary *cwnd* backoff when the reverse path is congested. Congestion in the reverse path is not caused by the aggressiveness of the sender, and decreasing *cwnd* will not improve congestion.

Some delay-based CC algorithms attempt to use only the OWD along the forward path (direction of data flow), to avoid reacting to congestion in the reverse path. One method of OWD calculation is to timestamp packets before transmission. At the receiver, the packet timestamp is extracted and subtracted from the local time. The receiver attaches the calculated OWD to the ACK reply packet.

A problem with this approach is that the direct calculated OWD will not have any meaning without strict time synchronisation between the sender and the receiver which is difficult to achieve. However, if the CC algorithm computes the difference between the calculated OWD and OWD_{base} (OWD_{min}), then the difference will be the one way queuing delay without time synchronisation.

$$OWD_q = OWD - OWD_{base}$$

The downside of using OWD in CC is that it requires receiver side modification, making deployment of such CC algorithms more challenging.

An alternative is to use the TCP timestamp extension [50] to calculate OWD. The sender subtracts the TS Echo Reply from the TS Value fields in the ACK packet to estimate the forward OWD. However, TCP timestamp is an optional feature which not all stacks implement or enable by default. Also, OWD measurement can suffer from clock drift between the sender and the receiver, leading to an increase or decrease of OWD estimation over time. This issue can be addressed by either resetting the OWD_{min} measurement regularly, or by using methods to estimate clock drift [6], [51].

Generally, the *delay signal* is used to estimate the queuing delay D_q for the bottlenecks along the path between the source and the destination. As a bottleneck queue fills, the packet queuing time is translated to latency.

Most delay-based CC algorithms calculate D_q from the path delay by estimating the fixed base (or propagation) delay D_{base}. D_{base} is practically determined as the smallest delay D_{min} seen during a period of time and assumes that the queues along the path completely drain (no queuing delay) at some points during that period. Then, D_q is estimated as $D_q = D - D_{base}$ where D is the measured delay.

Despite the potential advantages of using the *delay signal*, it comes with a number of difficulties. Primarily, the assumption that D_{base} equals D_{min} is not always true, which can lead to over or under-estimation of D_{base}.

Over-estimation of D_{base} leads to an under-estimation D_q and can result in non-detection of a congested network state. Alternatively, under-estimation of D_{base} can result in false detection of network congestion. Errors in measuring D_{base} are caused by persistent queues in the bottleneck. The aggressive nature of loss-based CC algorithms are an obvious source of standing queues in a heterogeneous environment, however standing queues can be formed by delay-based CC algorithms as well. Most delay-based CC algorithms try to achieve optimum throughput, requiring *cwnd* to be at least BDP (see section II-A). This results in queues always being partially filled. These errors can impact on CC performance where algorithms can be either over or under-aggressive, leading to to problems such as unfairness, the *latecomer advantage*, and generation of persistent large queues.

The latecomer advantage problem describes the case where a new flow gets higher bandwidth share (higher throughput) through a congested bottleneck than preexisting flows [52]. This problem is common in threshold delay-based CC algorithms that aim to maintain a constant number of packets in the queue. For example, Figure 9 plots *cwnd* versus time for the staggered starts of three LEDBAT flows (section V-A7) sharing a bottleneck. The plot illustrates how the new flows increase their *cwnd* (achieving higher throughput) while the existing flows decrease their *cwnd* (achieving lower throughput).

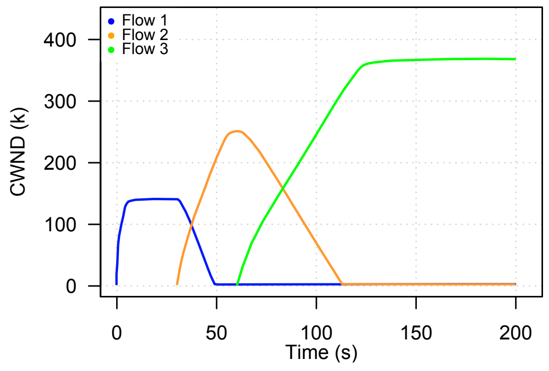


Fig. 9. *cwnd* vs time illustrates the latecomer advantage problem

This is due to each new flow measuring a higher D_{min} due to existing standing queues in the bottleneck. This leads to overestimating D_{base}, allowing the new flow to be more aggressive in increasing *cwnd*. At the same time, existing flows decrease their own *cwnd* as they interpret the new flow's aggressiveness as network congestion.

A second challenge is the assumption that the path delay is unchanged during a connection's life time, or at least over a period of time. However, this assumption is also not always true (for example, due to path rerouting) [53], [54].

Another challenge when using *delay signal* is the noisiness of delay measurement due to variation in queue occupancy and network jitter. The noise is exacerbated under a heavy load environment and weakens the correlation between the sampled delay signal and congestion [49].

The noise can be reduced using a filter, such as the exponentially weighted moving average (EWMA) filter [51]. However, filtering the signal may reduce the responsiveness of delay-based CC.

One final issue when using *delay signal* is delayed ACKs. The TCP delayed ACK option is used to reduce the number of ACK packets in the reverse path to reduce resource wastage. Delayed ACKs can cause inaccurate D_q estimation if the ACK is incorrectly matched to the corresponding data packet.

Instead of treating the delay as a pure signal (threshold), the delay-gradient can be used to infer congestion. Use of the delay gradient was first proposed by Jain in the CARD CC algorithm which uses the normalised delay-gradient of RTT to detect congestion [55].

Hayes et al. [10] proposed a new algorithm that utilises the average smoothed delay-gradient $\bar{g_n}$ of RTT_{min} and RTT_{max} seen in the measured interval to estimate congestion level. Using the gradient of minimum and maximum RTT, and the average smoothed filter, reduces the noisiness of the RTT gradient. Depending on $\bar{g_n}$ sign and magnitude, CC algorithm can increase or decrease *cwnd*. Using this signal it is possible to distinguish between packet loss related to congestion and loss related to a noisy environment such as a wireless network.

In addition to using the *delay signal* in congestion detection, some CC algorithms use this signal in calculating *cwnd*. Moreover, it is worth noting that not only delay-based TCP CC algorithms utilise the delay signal but also other transport protocols. For example, LDA+ [56] and MLDA [57] rely on the delay measurement in additional to loss signal to control sending rate of Real-time Transport Protocol (RTP) [58] and UDP multicast respectively.

B. Explicit Congestion Feedback Signals

Explicit congestion feedback refers to explicit signals sent by the bottleneck to inform the end-host of the congested state of the bottleneck. The sender's CC algorithm should respond to the signal by reducing *cwnd*. Bottlenecks use different mechanisms to detect congestion in their buffers and mark packets when congestion is encountered. It is clear that *explicit feedback* signals needs cooperation from the bottleneck, network protocol and transport protocol in order to function.

Explicit feedback for TCP/IP is typically implemented using the Explicit Congestion Notification (ECN) extension [3]. In the forward path, ECN supports packet marking, using dedicated bits in the IP header to encode congestion state.Additionally, ECN utilises bits within the TCP header which are used to inform the sender that congestion has happened and an action has been taken place.

If the router supports ECN, it marks the packet when the congestion is detected. When processing the marked packet, the receiver sets a flag in the ACK packet. Upon receipt of an ACK with congestion flag set, the sender reduces its *cwnd*.

To support ECN, a bottleneck router needs to detect and signal congestion before complete exhaustion of available buffer space, such as using AQM in place of the traditional Droptail mechanism (as described in section III). The AQM can then mark ECN-capable IP packets when congestion is experienced (rather than drop them).

As originally proposed [3], TCP was expected to react to an ECN signal in the same way as to packet loss (e.g. halving *cwnd* for TCP Reno). However, a new proposal suggests that *cwnd* should be reduced by a smaller amount for an ECN signal than for packet loss as the ECN signal is likely generated by an AQM-enabled bottleneck emulating a small queue [59]. This new proposal can improve TCP throughput without causing network collapse.

While using *explicit feedback* for CC reduces packet loss and can improve overall network performance, there are some difficulties. Middleboxes, including old, non-ECN aware firewalls, intrusion detection systems, and load balancers, may respond with an RST (reset connection) packet or drop the packet silently when processing a packet with the ECN enable flags set.

Another issue involves non-compliant hosts which pretend to support ECN during connection negotiation but do not respond to marked packets. This results in the receiver attaining higher throughput than other flows sharing the bottleneck, and increased congestion in the bottleneck.

This vulnerability can be addressed in the AQM by dropping packets instead of marking them when congestion exceeds a specific threshold. For example, PIE has a safeguard against such behaviour by dropping ECN enabled packets when queuing delay becomes high, and the sender will reduce *cwnd* as a reaction to packet loss. An alternate solution is to segregate flows into queues using scheduling techniques such as FQ-CoDel or FQ-PIE (section III-B), thereby isolating well-behaved flows from unresponsive flows.

ECN-based CC approaches rely on explicit congestion feedback to estimate congestion intensity and react (by reducing *cwnd*) in different degrees based on that intensity. The purpose of this type of CC is to provide a low-latency and high-throughput protocol with low loss rate for controlled network environments such as data centres. Unlike AQM configured for conventional TCP, ECN-based CC algorithms require AQMs to mark packets much earlier (having lower target delay/occupancy and very short burst tolerance) to provide very low latency. The throughput of ECN-based CC will not be affected by such very shallow buffer emulation since *cwnd* back-off factor is adaptive based on congestion intensity. A well-known example of ECN-based CC approaches are DCTCP [60], Deadline-Aware Data Center TCP (D2TCP) [61] and L2DCT [62]. It is worth noting that a recent study proposes an architecture called L4S [63] to use ECN-based CC on the Internet by utilising a special type of AQM (e.g. DualQ Coupled AQM [64]). The proposed AQM separates classic TCP flows from ECN-based flows in two different queues and then priority packet scheduler is used to provide fair bandwidth share.

C. TCP Congestion Control Algorithms

Standard TCP deploys a combination of techniques (slow start (SS), congestion avoidance (CA), fast retransmit and fast recovery) to respond topacket losses [21]. SS is used to probe the link capacity when no previous information is available about the link. CA aims to prevent heavy network congestion while adapting to changes in network conditions. Fast retransmit and fast recovery are used to quickly resend the missing packets and recover from theses losses. In this section, we describe these algorithms in some detail as they are considered the base for most TCP variants.

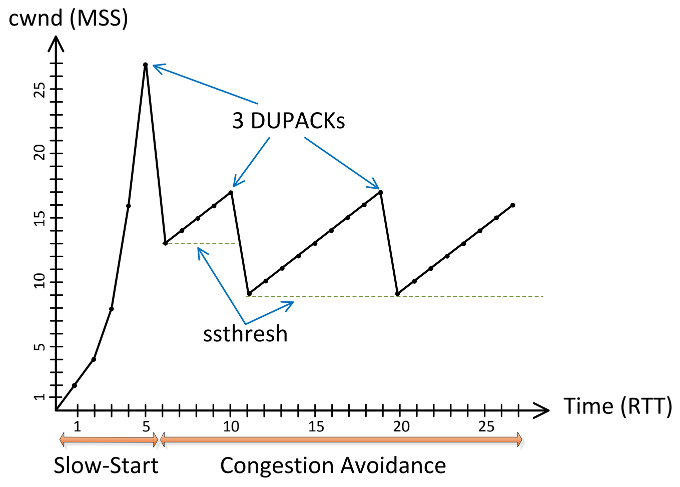
1) Slow Start: In the absence of specific network feedback, a TCP sender is typically unaware of path capacity when a connection is first established. TCP uses the slow start algorithm to initially probe a path's capacity.

To begin a new connection in SS mode, TCP sets *cwnd* to the Initial Window (IW) and transmits this IW of bytes. For each received ACK that acknowledges new data, *cwnd* is increased by no more that one MSS, typically doubling *cwnd* every RTT. This mode is referred to as *slow* start because the sender does not begin with *cwnd* set to some large (and potentially excessive) initial value.

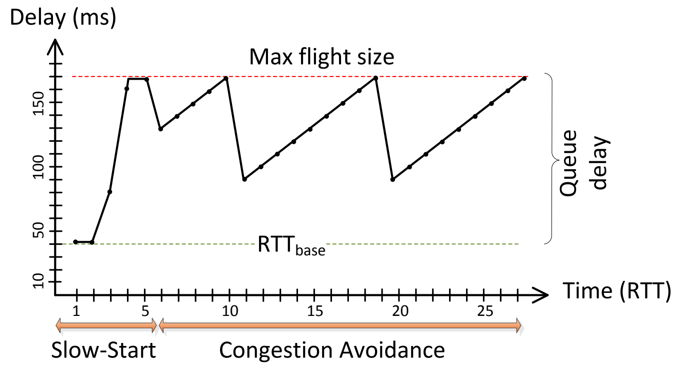
TCP exits SS and enters congestion avoidance mode when congestion is detected. Path capacity is estimated as a portion of *cwnd* that was realised when the congestion was detected.

To distinguish between SS and CA modes, TCP maintains a state variable called slow start threshold (*ssthresh*). The SS algorithm runs when $cwnd < ssthresh$, otherwise the CA algorithm is executed. Initially, *ssthresh* is set to a high value to allow the SS algorithm to probe available bandwidth quickly. Following each congestion event, *ssthresh* is set as a multiple (usually half) of *cwnd*.

Figure 10 plots *cwnd* progression and RTT versus time for TCP Reno. The first six time samples cover the SS phase of the flow. The exponential growth of *cwnd* is shown in Figure 10a where *cwnd* increases to about 27 MSS at time 5RTT. In Figure 10b, we also see that RTT increases above the base RTT of 40ms after time 2RTT once *cwnd* becomes larger than the path BDP. RTT continues to increase to about 170ms when the bottleneck queue is full and packets are dropped.



(a) cwnd vs time



(b) RTT vs time

Fig. 10. TCP Reno Slow-Start and Congestion Avoidance

When packet loss is used to detect congestion, the exponential growth of *cwnd* can lead to a large overshoot of the optimum value. This can result in high packet loss within one RTT, leading to waste in network bandwidth, long unresponsive periods and increased loads on the end-host operating systems during the loss recovery period [65]. This problem is exacerbated in networks with large BDPs network as *cwnd* grows to a large size and there is an increased time period before congestion feedback is noticed by the sender.

Consequently, improvements have been proposed to find a safe exit point from SS without resulting in high packet loss and low bandwidth utilisation. One proposed algorithm is Hybrid Start (HyStart) [65] which uses the ACK trains technique and sampled RTT to find a safe exit point to CA.

2) Congestion Avoidance: During congestion avoidance (CA), TCP CC tries to avoid congestion by increasing *cwnd* slowly during periods of no congestion, and reducing it significantly when congestion is detected. Some algorithms try to stabilise *cwnd* when they infer that the available bandwidth is fully utilised.

The standard technique for maintaining *cwnd* is the Additive Increase/Multiplicative Decrease (AIMD) algorithm, where *cwnd* increases linearly by α once per RTT and multiplicatively decreases it by β (where α and β are algorithm specific constants). For example, TCP Reno uses $\alpha = 1$ and $\beta = 0.5$ [21], resulting in *cwnd* increasing by no more than one MSS bytes per RTT, and being halved when congestion is detected.

In order to keep $cwnd \geq BDP$ and achieve full link utilisation (see Section II-A), *cwnd* should be greater than $2 \times BDP$ upon packet loss. This can be achieved only if the bottleneck buffer size equals at least the BDP of the connection. Otherwise, after backoff *cwnd* will be dropped to less than BDP and require multiple RTTs in CA mode to regrow back to BDP. This can cause severe degradation of throughput in large BDP networks.

To simplify implementation, the additive increase can be performed using the appropriate byte counting method. The number of acknowledged bytes is accumulated until they become greater than *cwnd*, and then *cwnd* is increased by one MSS. Another formula that can be used to update *cwnd* is given in equation 1.

$$cwnd_{i+1} = cwnd_i + MSS * \frac{MSS}{cwnd_i} \qquad (1)$$

When congestion is detected via RTO firing, TCP resets *cwnd* to one MSS and sets *ssthresh* to no more than half of flight size. The missing packets are then resent and TCP reenters SS.

When congestion is detected via 3DUPACK, TCP enters the fast retransmit and fast recovery phase.

3) Fast Retransmit and Fast Recovery: TCP Tahoe enters a *fast retransmission* phase when packet loss is detected via 3DUPACK. The missing packet is immediately retransmitted, *ssthresh* is set to half of *cwnd* and *cwnd* is reset to one MSS. TCP then enters SS.

TCP Reno augments the response to 3DUPACKs with fast retransmission and *fast recovery* [21]. The missing packet is immediately retransmitted, *ssthresh* is set to half of *cwnd* and *cwnd* is set to *ssthresh* plus $3 \times MSS$. This inflates the congestion window to reflect the three packets that departed the host after the missing packet. *cwnd* is subsequently incremented by one MSS for each additional duplicate ACK received as a reflection of the additional packets delivered to the destination. When new data is ready to be sent, TCP should send one MSS worth of bytes if *cwnd* allows.

Fast recovery finishes by receiving a new ACK that acknowledges unacknowledged data. After that, TCP Reno sets *cwnd* to *ssthresh*, and enters CA.

TCP Reno fast recovery is inefficient when multiple packets losses occur in the same transmission window since the cumulative acknowledgement doesn't reflect losses after the first missing packet. A new fast recovery algorithm called NewReno was proposed to address this issue [25].

Another solution to multiple losses within a window is to use the TCP Selective Acknowledgment (SACK) option [26]. SACK allows the receiver to inform the sender the exact sequence of bytes that have been received so the sender need only resend the missing segments without waiting for multiple RTTs. The number of SACK blocks (range of received bytes) within one packet is limited by the TCP options field. As a consequence, SACK may not be able to provide all received byte ranges if many non-contiguous losses occur within one window.

D. Congestion Control Metrics

It is important to understand congestion control evaluation metrics to be able to study and compare the performance of different CC algorithms. Various CC algorithms are designed with different aims and working environments. Some protocols focus on improving specific metrics while others focus on trading-off multiple metrics. The main metrics for evaluating congestion control mechanisms are [66]:

- Throughput: The amount of data sent per time interval. Can be measured for routers as aggregate link utilisation, for flows as connection transfer times, and for users as user wait times.
- Delay: Measures the additional queuing delay caused by the CC algorithm.
- Packet Loss Rate: Measures wastage of network resources.
- Fairness: The degree of equality in resource allocation.
- Convergence time: The time required to for flows to converge to fairness.

CC algorithms often need to consider a trade-off between metrics. For example, loss-based CC typically has higher throughput at the expense of increased delay and packet loss. Alternatively, delay-based CC has improved delays and packet loss at the expense of lower throughput when competing with loss-based flows. To achieve both high throughput and low queuing delay, CC schemes aim to maximise the *power* metric [67]. Power metric is given in Equation 2 where x is flow's throughput, RTT is the current round trip time and α is a constant. If $\alpha > 1$, power metric will give a preference to throughput over the response time of the flow (higher throughput, higher queuing delay). If $\alpha < 1$, this metric gives a preference for the response time (lower throughput, lower queuing delay).

$$power = \frac{x^{\alpha}}{RTT} \tag{2}$$

The choice of CC algorithms and preferred performance metrics are influenced by application requirements. However, in environments that include a mixture of loss and delay based CC, the ideal outcome is typically unachievable. Pure delay-based CC can suffer from unfair resource allocation as they typically defer to loss-based CC due to the high queue occupancy caused by loss-based CC.

Ensuring fair capacity sharing is not a trivial task, especially when different CC algorithms coexist over the same path, or when flows travels over different path distances. One of the most commonly used indices for measuring resource sharing fairness is Jain's fairness index [68] [66]. Jain's index is given in Equation 3 where n is number of flows and x_i is the throughput of the *i*th flow. This index ranges from 0 to 1, and is at a maximum when all flows receive the same allocation.

$$fairness = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2} \tag{3}$$

When a new flow joins a shared bottleneck, bandwidth should be reallocated for all competing flows. Convergence time in a high BDP environment is important as large amounts of data can be transferred over short time intervals.

One measurement of convergence time is the delta-fair convergence time [69]. This measures the time taken for two flows to go from a fully unfair share of the link capacity, to having near fair sharing of link capacity. The time is calculated as per Equation 4, where B is the total bandwidth, δ is the fairness wants converge to and b_0 is the initial bandwidth allocated to the new flow.

$$\delta - fair\,conv = (B - b_0, b_0) \;\rightarrow\; (\frac{1+\delta}{2}B, \frac{1-\delta}{2}B) \tag{4}$$

In addition to these metrics, robustness to noisy environments, misbehaving users, minimising *cwnd* oscillations and dependability are considered important in many cases.

V. CC ALGORITHMS THAT UTILISE THE DELAY SIGNAL

We propose a taxonomy (Figure 11) to categorise the behaviour of TCP Congestion Control Algorithms with respect to their use of different congestion signals. Within this taxonomy we define primary categories of: 1) ECN-Based which use explicit congestion notification as described in Section IV-B; 2) Loss-Based which primarily rely on packet loss as a congestion signal; 3) Delay Based which primarily rely on the delay signal; 4) Hybrid which use a combination of both Loss and Delay signals; and 5) Bandwidth Estimation Delay Sensitive which use link capacity estimation and delay measurements to regulate the sending rate.

We further sub-classify Loss Based algorithms into Pure Loss Based approaches and Delay Sensitive algorithms to differentiate those that may occasionally use the loss signal to achieve their aims. We also sub-classify hybrid algorithms into Dual Signal and Dual mode algorithms.

Dual signal approaches utilise both loss and delay signals. The delay signal allows the CC algorithm to scale quickly without stressing the network. They are usually deployed in large BDP networks where traditional loss-based approaches can be slow to achieve high link utilisation.

Dual mode approaches alternate between using loss and delay signals based on internal state. They typically use the delay signal to infer early congestion and better manage queue latency, and revert to using the loss signal when competing with loss-based flows to achieve reasonable inter-flow fairness.

Pure loss based and ECN-based approaches are out of scope for the rest of this paper.

In general, CC algorithms that utilise the delay signal use RTT or OWD based metrics to detect the degree of network congestion. Although different CC algorithms may use customised metrics, common delay metrics are queuing delay, queue occupancy or delay gradient. Algorithms that estimate queuing delay attempt to keep latency under a predefined time threshold. Algorithms that estimate queue occupancy attempt to keep bottleneck buffer utilisation to a specific threshold (bytes or packets) or to perform early detection of congestion events. Delay gradient algorithms avoid the use of thresholds to avoid issues around RTT and base RTT estimation.

A. Delay-based Algorithms

Most delay-based CC algorithms are *threshold-based* which infer early congestion in the network when the measured delay signal exceeds a pre-configured or dynamically calculated threshold or thresholds. A few use *delay-gradient* approaches to infer network congestion by monitoring congestion trends in bottleneck buffer and make decisions based on the rate of change of queuing delay.

Most delay-based CC algorithms aim for high link utilisation with short bottleneck queues. However, others are designed for background bulk file transfer applications, and typically aim is to achieve a lower bandwidth share when competing with standard flows. Delay-based CC can achieve high throughput by reducing the oscillatory *cwnd* behaviour of standard TCP by slightly reducing the window size when the queuing delay reaches a defined threshold.

Unfortunately, CC algorithms of this category typically suffer from unfair resource allocation when sharing a bottleneck with loss-based CC and experience the latecomer advantage problem described in Section IV-A2. Table I summarises the properties of the delay-based TCP variants reviewed in this section.

1) TCP DUAL: In 1992, Wang and Crowcroft [70] proposed an enhanced algorithm, called TCP DUAL, to minimise the oscillation of TCP Tahoe's *cwnd* dynamic in CA mode. Dampening these oscillation helps reduce fluctuations in buffer utilisation that can lead to RTT instability and periodic packets losses.

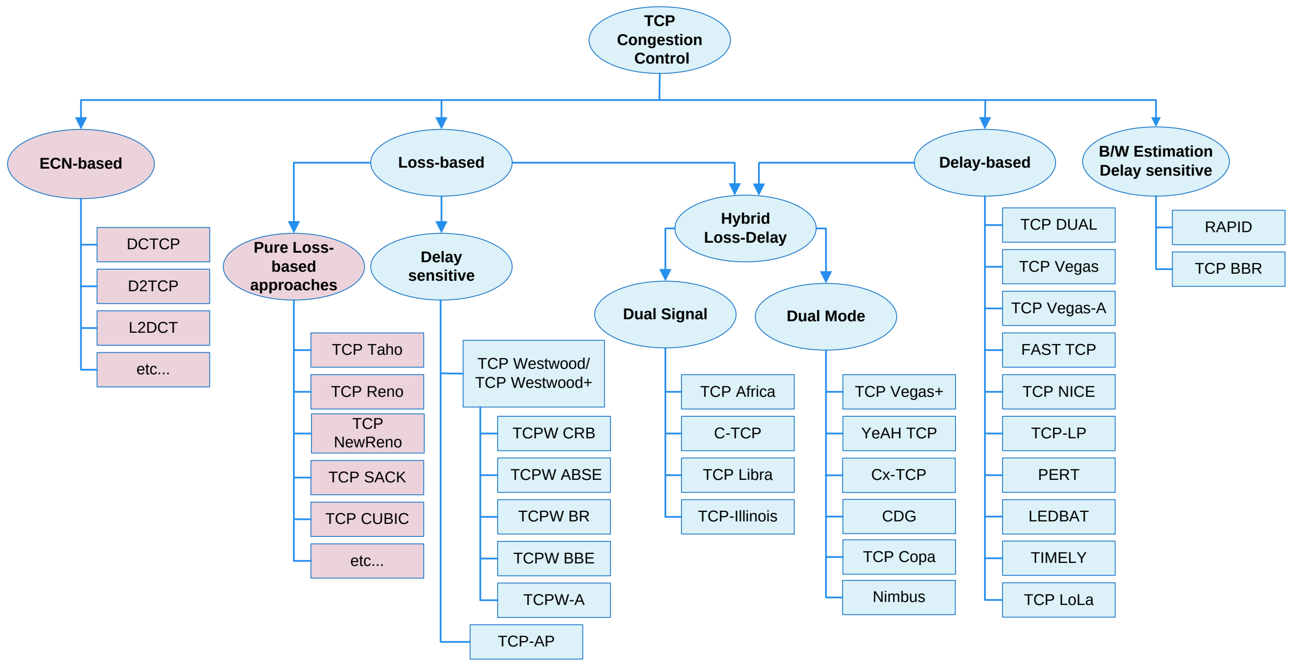


Fig. 11. Taxonomy of TCP congestion control techniques reviewed in this survey

TABLE I
DELAY-BASED TCP VARIANTS REVIEWED IN SECTION V-A

TCP variant	Section	Algorithm aims	Delay signal type	Metrics
TCP DUAL [70]	V-A1	minimise *cwnd* oscillation of TCP Tahoe, better throughput	RTT	queuing delay
TCP Vegas [22]	V-A2	minimise *cwnd* oscillation, low queuing delay, better throughput	RTT	queue occupancy
TCP Vegas-A [53]	V-A3	remedy TCP Vegas path re-routing and fairness issues	RTT	queue occupancy
FAST TCP [71], [72]	V-A4	scalability in large BDP networks, low queuing delay	RTT	queue occupancy
TCP NICE [7]	V-A4	low priority, low queuing delay	RTT	queue delay
TCP-LP [51]	V-A5	low priority, low queuing delay	OWD	queue delay
TCP PERT [73]	V-A6	low queuing delay	RTT	queue delay
LEDBAT [6]	V-A7	low priority, low queuing delay	OWD	queue delay
TIMELY [74]	V-A8	low queuing delay in datacentre environments	RTT	delay gradient
TCP LoLa [75]	V-A9	scalability in large BDP networks, low queuing delay	RTT	queue delay

TCP DUAL estimates the queuing delay using RTT measurement to indicate the network congestion level and reduces *cwnd* before a packet loss happens. The algorithm assumes that the base RTT is RTT_{min} , and RTT_{max} represents the RTT of the highest congestion level along the path. RTT_{min} and RTT_{max} measurements are reset whenever RTO is triggered. At any time, the two-way queue delay (Q_i) can be estimated shown in equation 5.

$$Q_i = RTT_i - RTT_{min} \tag{5}$$

DUAL attempts to keep the queuing delay at a point between the minimum and maximum queue delay ($Q_i \rightarrow \delta \times Q_{max}$) where $Q_{max} = RTT_{max} - RTT_{min}$ and $0 < \delta < 1$ ($\delta = 0.5$ is used in [70]). In other words, it attempts to keep the RTT close to a threshold th where $th = (RTT_{max} + RTT_{min}) \times \delta$.

On every other received ACK, if $RTT_i > th$, TCP DUAL multiplicatively decreases *cwnd* by 7/8 to fine tune the RTT around th. If network congestion is detected using RTO mechanism, TCP DUAL behaves similarly to TCP Tahoe i.e. *ssthresh* = *cwnd*/2, *cwnd*=1 and moves to SS phase. If no congestion is detected using the delay or loss signals, TCP DUAL increases *cwnd* by one MSS every RTT.

Due to the delay component of TCP DUAL, this algorithm is affected by the latecomer advantage unfairness problem. Additionally, in common with other delay based back-off schemes, TCP DUAL flows can suffer from low bandwidth sharing when competing with loss-based flows.

2) TCP Vegas : An early, well known delay-based TCP CC, TCP Vegas [22] aims to achieve maximum throughput, low packet loss and queuing delay, with minimum *cwnd* oscillation. It uses *cwnd*, the current RTT and RTT_{base} (derived from the minimum witnessed RTT) to regularly estimate the number of in-flight bytes that reside in the bottleneck buffer, while aiming to keep this number small.

TCP Vegas deploys an Additive Increase Additive Decrease (AIAD) approach when congestion is controlled using the delay component. *cwnd* is increased by at most one MSS every RTT, ensuring the algorithm is not more aggressive than standard TCP. When packet loss is detected, TCP Vegas mimics standard TCP by halving *cwnd*.

During CA, TCP Vegas calculates the difference between the expected and actual sending rate to estimate the data currently queued at the bottleneck. The expected sending rate can be calculated as the throughput when no congestion is present in the bottleneck i.e. the RTT equals RTT_{min}:

$$expected\,rate = \frac{cwnd_i}{RTT_{min}}$$

The actual sending rate is the calculated based on the actual RTT (RTT_i) :

$$actual\,rate = \frac{cwnd_i}{RTT_i}$$

To estimate the number of queued packets at the bottleneck Δ, the difference between the rates is multiplied by RTT_{min}:

$$\Delta = (expected\,rate - actual\,rate) \times RTT_{min} \tag{6}$$

TCP Vegas calculates Δ upon receipt of each ACK and compares Δ with algorithm parameters α and β (default $\alpha = 1$ and $\beta = 3$). These parameters control the length and stability of the bottleneck queue. The protocol aims to maintain the queue length between α and β.

When $\Delta < \alpha$, TCP Vegas increases *cwnd* by one segment during the next RTT. When $\Delta > \beta$, network congestion is inferred and *cwnd* is decreased by one segment during the next RTT. Otherwise *cwnd* is left unchanged.

The rationale of this algorithm is that if a sender can send a *cwnd* worth of data without observing a large increase in RTT, this means the link is under-utilised and we can increase *cwnd*. Alternatively, if the increase in RTT is such that Δ exceeds threshold β, this means the link is over-utilised and we should reduce *cwnd*.

TCP Vegas also proposes an enhancement to the TCP Reno SS mechanism to detect the available bandwidth and exit SS before packet loss occurs. Specifically, it exits SS when $\Delta > \alpha$ where Δ is calculated as above upon receipt of every other ACK. TCP Vegas also introduces a more timely technique to detect loss before receiving the third duplicate ACK.

Although these modifications aim to reduce the stress on the network, experimental results [22] show a very small impact on overall network performance due to the short working time of SS and fast retransmission compared with CA.

Barkmo and Peterson [22] claim that TCP Vegas is able to achieve 37% to 71% better throughput and reduction in packet losses by 1/5 to 1/2 than TCP Reno on the Internet. Unfortunately, later studies [23], [76], [77], [78], [79] demonstrate a number of issues with TCP Vegas including 1) low fair share of bandwidth when competing with Reno-style flows (loss-based CC); 2) low throughput following sudden increases in base RTT (eg. path rerouting); and 3) latecomer advantage (section IV-A2) due to incorrect base RTT estimation.

3) TCP Vegas-A: Despite the limitations, TCP Vegas still displays desirable characteristics. Srijith et al. [53] claim that the re-routing and fairness problems when sharing a bottleneck with Reno flows can be remedied by dynamically adapting α and β coefficients based on the actual sending rate. They propose a modification to TCP Vegas called TCP Vegas-A.

Vegas-A uses the default α and β values (1 and 3 respectively) at the start of the connection, and keeps these values as the minimum boundaries. After that, Vegas-A dynamically changes these values based on the network conditions.

When $\alpha < \Delta < \beta$, the algorithm is in steady state. Vegas-A attempts to probe the available bandwidth to adjust α and β to maximise throughput. When Vegas-A detects an increase in the actual rate, α, β and *cwnd* are incremented.When $\alpha > 1$, $\Delta < \alpha$ and there is a decrease in the actual transmission rate, Vegas-A assumes that the coefficients have been over-estimated and decreases α, β and *cwnd*. When $\alpha > 1$, $\Delta < \alpha$ and there is an increase in the actual transmission rate, *cwnd* is increased. Otherwise, the algorithm adjusts *cwnd* using the TCP Vegas rules.

Using ns-2 simulations, Srijith et al. [53] show overall improvement with TCP Vegas-A compared to TCP Vegas for the path re-routing issue over both wired and fluctuating RTT satellite links. They also show better fairness when Vegas-A competes with TCP Reno flows.

However, Vegas-A is still unable to obtain fair capacity sharing with Reno-style flows when the number of flows is small. Further, when the number of TCP Vegas-A flows becomes relatively high, the fairness problem inverts and the Vegas-A flows get a higher capacity share than Reno flows.

4) FAST TCP: Inspired by the TCP Vegas idea of controlling congestion based primarily on the delay signal, Jin et al. [71], [72], proposed FAST TCP, targeting low queuing delays and high bandwidth utilisation in large BDP paths.

Similar to TCP Vegas in the steady state, FAST tries to maintain a fixed number of packets (α) in the bottleneck queue by using RTT_{base} (derived from observed RTT_{min}) and the current average RTT. Instead of adjusting *cwnd* by one MSS every RTT interval, TCP FAST updates *cwnd* every fixed interval (eg. 20ms) using the specialised equation:

$$w_{i+1} = min\left\{2.w_i,\, (1-\gamma).w_i + \gamma\left(\frac{RTT_{min}}{RTT_i}w_i + \alpha\right)\right\}$$

The window smoothing factor (γ) is a configurable parameter between 0 and 1 that affects the window update response to congestion. The target number of packets in the queue (α) is a constant that controls the protocol fairness.

Selection of α is an open challenge but the authors of FAST TCP used large values (eg. 200) in their experimental evaluation [72]. The window adjustment step is large when the the current RTT is close to the base RTT, and small as the protocol approaches the steady state (α packets buffered in the queue).

FAST TCP also uses packet pacing to control the burstiness of the congestion window mechanism in a large BDP environment and to provide accurate RTT measurement.

Emulated network and simulation based experiments show good overall throughput, scalability, stability, RTT-, inter- and intra-fairness for FAST TCP [71], [72].

However, Tan et al. [80] show unfairness problems and large variation in queue occupancy related to inaccurate propagation delay estimation, as for TCP Vegas. This inaccurate estimation happens during route change and standing queue scenarios. They also find that the FAST TCP *cwnd* update rule is more aggressive than standard TCP which can cause unfairness in specific scenarios.

Unlike most congestion control algorithms, FAST TCP is a commercial CC algorithm and is protected by patents [81], [82]. It is one of the few delay-based algorithms that is actually used in practice over the Internet.

TCP NICE

TCP Nice [7] is a scavenger class CC algorithm based on TCP Vegas with a more sensitive congestion detection mechanism. TCP Vegas by itself provides low priority CC due to its proactive reaction to congestion, but not low enough to be scavenger class CC.

During one RTT interval, TCP Nice counts the number of times that the estimated bottleneck queuing delay is greater than a fraction of the maximum queuing delay. In other words, the number of times measured RTT is larger than $RTT_{min} + (RTT_{max} - RTT_{min}) \times threshold$), where $threshold$ defines the target fraction.

If the count is greater than a *fraction* of the congestion window, TCP Nice halves *cwnd*, otherwise it behaves like TCP Vegas. When loss is detected, TCP Nice halves *cwnd*.

Another mechanism used to ensure low priority is that *cwnd* is allowed to decrease to a value lower than one, this postpones packet delivery for a number of RTTs.

Although TCP NICE flows realise low throughput when competing with Reno-style flows, there is a concern about how well NICE can utilise the available capacity when just LPCC flows exist. As with TCP Vegas, NICE also experiences the latecomer advance problem due to incorrect base RTT estimation.

5) TCP-LP: TCP-LP [51] is an LPCC algorithm that aims to utilise available bandwidth without affecting foreground TCP flows. Unlike TCP NICE, TCP-LP uses EWMA smoothing of one-way delay measurements to infer queuing delay in the forward path, avoiding delay fluctuations caused by reverse path traffic.

TCP-LP uses the TCP timestamps option [50] to calculate a form of OWD as the difference between receiver's timestamp in the ACK packet and the sender timestamp copied to the ACK. Without synchronised clocks this is not a true OWD (section IV-A2), so TCP-LP utilises the minimum and maximum measurements to calculate one-way *queuing* delay.

TCP-LP infers early congestion in the forward path when equation 7 is true (a similar strategy to that used by TCP Nice), where we are measuring if queuing delay is greater than a fraction of the maximum queuing delay.

$$OWD_i > OWD_{min} + (OWD_{max} - OWD_{min}) \times th \quad (7)$$

TCP-LP reduces *cwnd* more aggressively than standard TCP when congestion is detected. At the first sign of congestion, *cwnd* is halved. If another congestion event occurs within one RTT, TCP-LP infers that persistent congestion exists in the network and subsequently sets *cwnd* to one MSS. During periods of no congestion, *cwnd* is increased similar to TCP Reno.

Using ns-2 simulation and a real-world Linux implementation, the authors of TCP-LP [51] show that TCP-LP achieves its design goals of yielding available bandwidth to competing standard TCP flows, and high bandwidth utilisation with good fairness when no high priority flows are competing.

It is unknown how well TCP-LP will achieve its goals in an AQM or wireless environment. More evaluation is required for this and similar techniques in different scenarios.

6) PERT: Probabilistic Early Response TCP (PERT) [73] is a delay-based algorithm that emulates AQM and the end-host without modification to the bottleneck. PERT authors claim that any AQM can be emulated at the end-hosts but they choose RED [32] and PI [83] AQM.

PERT uses a probabilistic back-off function ($P_{backoff}$) based on delay measurements at the end host. On every receiving ACK, PERT smooths the instantaneous RTT_i sample using Exponentially Weighted Moving Average (EWMA) to produce $SRTT_i$ to reduce signal noise. Then it uses $SRTT_i$ and RTT_{min} to calculate back-off probability.

PERT defines three thresholds th_{min} (defaults to RTT_{min}+ 5*ms*) , th_{max}(defaults to RTT_{min}+ 10*ms*) and P_{max} (defaults to 0.05). By using RTT_{min}, PERT estimates instantaneous queuing delay similar to TCP DUAL. $P_{backoff}$ is zero if $SRTT_i$ is less than th_{min}. $P_{backoff}$ increases linearly until it reaches P_{max} when $SRTT_i$ equals th_{max}. Then, $P_{backoff}$ increases faster to reach one when $SRTT_i$ becomes larger than or equal $2.th_{max}$. Figure 12 shows the back-off probability function that PERT uses.

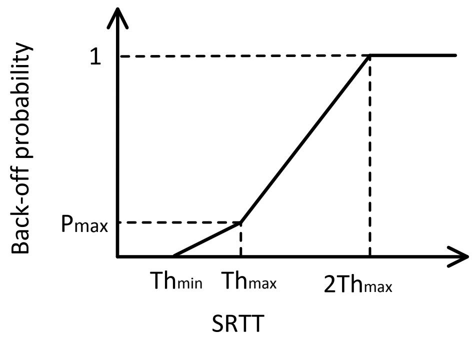


Fig. 12. PERT back-off probability function

PERT uses 0.65 multiplication decrease factor if the congestion is detected using the delay component and 0.5 if packet loss occurs. Additionally, it responds to congestion once every RTT since the effect of back-off is not observed until after an RTT. This reduces the number backing off times per congestion event.

Using ns-2 network simulator[84] and fluid mathematical model, PERT authors [73] find that this algorithm achieves very low queuing delay and all most no packet loss with good fairness between competing PERT flows. They also find that PERT is able to utilise the link in similar way as using AQM on the bottleneck. However, PERT does not solve the coexisting with loss-based flows problem and PERT authors highlight some possible solutions as future work.

Kotla et al. [85] propose a modification to PERT to allow better coexistence with loss-based flow by increasing the additive increase factor if high queueing delay is observed. However, it has been shown the this modification causes high loss rate and high queueing delay in many scenarios [86]. Without a functional coexistence mechanism, PERT cannot be deployed on the Internet since loss-based algorithms are the most widely used CC.

7) LEDBAT: Low Extra Delay Background Transport (LEDBAT) [6] is an LPCC that is widely implemented in different bulk transfer applications such peer-to-peer file transfer [87] and software updates. LEDBAT aims to keep the forward queuing delay relativity small to reduce interference with other flows, particularly flows used by latency sensitive applications such as Voice over IP (VoIP).

Similar to most delay-based CC, LEDBAT monitors queuing delay and considers an increase in that delay as an early signal of network congestion. By responding to this signal, LEDBAT flows defer to competing standard TCP flows.

As for TCP-LP, LEDBAT uses OWD measurement instead of RTT to avoid delay fluctuations in the reverse path.

LEDBAT utilises a predefined *target* threshold (default 100ms) for queuing delay. When no packet loss is detected, LEDBAT proportionally increases or decreases *cwnd* based on the relative difference between the *target* and estimated queuing delay (equation 8). If loss is detected, LEDBAT behaves like standard TCP by halving *cwnd*.

$$cwnd_{i+1} = cwnd_i + \frac{G \times MSS \times \Delta \times B_{acked}}{cwnd_i} \tag{8}$$

The gain scale (G) determines the *cwnd* growth/decline rate and should be no greater than one to ensure LEDBAT is not more aggressive than the standard TCP. B_{acked} is the number of newly acknowledged bytes. Δ is the normalised difference between the one-way queuing delay and *target* and is defined in equation 9.

$$\Delta = \frac{(target + OWD_{base} - OWD_i)}{target} \tag{9}$$

LEDBAT requires OWD measurement to be made for every packet transmitted by the sender in order to react accurately and quickly to changes in delay.

LEDBAT maintains a history (default ten entries) of base OWD where each element represents the measured OWD_{min} in a one minute interval. OWD_{base} is the minimum value of this list. The history is used to minimise the effect of sudden changes in base OWD estimation caused by delayed ACK, clock skew and re-routing problems.

Due to the low impact of LEDBAT flows on latency sensitive applications, BitTorrent, a very popular peer-to-peer file sharing protocol, uses this algorithm in its UDP-based transport protocol [88], [89]. Additionally, Apple Inc. implemented TCP-based LEDBAT to be used for sending operating system updates to their clients [9].

A number of studies has been evaluated the performance of LEDBAT [90], [54], [4]. These studies have found that LEDBAT introduces increasing delay due to measuring its self-induced delay, and suffers from issues related to incorrect propagation delay estimation (unfairness and latecomer advantage problems). Moreover, a study found that a special care should be taken when choosing LEDBAT parameters in AQM environments since *cwnd* backs-off will be controlled using loss signal as the delay will never reach LEDBAT target delay; otherwise LEDBAT flows become too aggressive [91].

8) TIMELY: TIMELY [74] is a rate-based delay-gradient CC algorithm optimised to function in a datacentre environment without the need for additional support from intermediate nodes such as network switches and routers. The authors state that TIMELY can achieve high throughput while maintaining low packet latency by relying on accurate RTT measurements for congestion detection.

RTT is typically very small in a datacentre environment, and software time-stamping is too inaccurate to to measure RTT. As such, the authors suggest using hardware time-stamping provided by modern Network Interface Cards (NICs) to obtain accurate microsecond resolution RTT measurements.

TIMELY uses a delay-gradient signal similar to CDG (section V-B3). Due to the high accuracy of the RTT measurements, the raw RTT can be used rather than RTT_{min} and RTT_{max}.

TIMELY also uses rate-based congestion control rather than a window-based mechanism. The *Rate Computation Engine* calculates the required sending rate from the delay-gradient signal, and the packet transmission is managed by the *Rate Control Engine.*

RTT is measured once every chunk of data (16KB - 64KB). TIMELY uses two threshold values for RTT of T_{low} and T_{high} specifying lower and upper bounds for acceptable RTT.

TIMELY infers the network is under-utilised if $RTT < T_{low}$; or $T_{low} < RTT < T_{high}$ and the normalised RTT gradient is negative. In this case the sending rate is additively increased by δ.

TIMELY infers the network is over-utilised if $RTT > T_{high}$; or $T_{low} < RTT < T_{high}$ and the normalised RTT gradient is positive. In the first case, the sending rate is reduced using equation 10. In the second case, the sending rate is multiplicatively reduced in proportion to the RTT gradient and β using equation 11.

$$rate = rate \times \left(1 - \beta \times \left(1 - \frac{T_{high}}{RTT}\right)\right) \tag{10}$$

$$rate = rate \times (1 - \beta \times normalized\_gradient) \tag{11}$$

The authors of TIMELY claim that this is the first delay-based CC algorithm to be used in a datacenter environment and is able to achieve high throughput and low latency without ECN support. They also found a strong correlation between

RTT and queue occupancy if an accurate measurement with proper sampling is used [74].

As TIMELY requires NIC hardware support, deployment is only possible where appropriate hardware is present. Further, Zhu et al. [92] found using fluid model and simulation that TIMELY converges to a stable point, but with arbitrary unfairness.

9) TCP LoLa: TCP LoLa [75] is another threshold delay-based congestion control algorithm that aims to achieve high link utilisation in long distance and high bandwidth networks while keeping the bottleneck queuing delay low. TCP LoLa endeavours to keep bottleneck buffer utilisation around a fixed target Q_{target} value regardless of the number of flows competing for bottleneck bandwidth. Moreover, it attempts to achieve fairness between flows having different path's RTT by using a proposed *Fair Flow Balancing* mechanism. In general, TCP LoLa is an enhanced TCP Vegas based algorithm with CUBIC *cwnd* growth function, better inter-flow RTT fairness and better RTT_{min} estimation.

More specific, TCP LoLa algorithm relies mainly on estimating the current two-way queuing delay Q_i using TCP DUAL method (Section V-A1, Equation 5). Similar to Vegas in SS phase, TCP LoLa uses queue occupancy based condition ($Q_i > 2.Q_{low}$) to exit SS before packets losses occur to prevent building-up long queue.

In CA phase, it uses the CUBIC TCP [42] algorithm to grow *cwnd* when Q_i is less than a predefined threshold Q_{low}. If $Q_i > Q_{low}$, LoLa enters fair flow balancing state to realise inter-RTT fairness. In this state, all flows sharing a bottleneck attempt to keep an equal number of bytes X in the bottleneck's buffer at the same time. X is calculated according to Equation 12 where ϕ is a constant and t is the difference between current time and time at entering the balancing state. The number of bytes in the buffer Q_{data} is estimated the same Vegas queue occupancy estimator (Equation 6). During the fair flow balancing if $Q_{data} < X(t)$, *cwnd* increases based on the difference between $X(t)$ and Q_{data}; otherwise *cwnd* is leaved unchanged.

$$X(t) = \left(\frac{t.1000}{\phi}\right)^3 \tag{12}$$

A flow exits the balancing state and enters *cwnd* holding state when $Q_i > Q_{target}$. In *cwnd* holding state, *cwnd* is kept unchanged for a certain amount of time (e.g. 250ms) to make all flows to return to normal operation state at the same time. After the holding time elapsed, *cwnd* decreases using a modified CUBIC function to realise a fully drained buffer. This allows flows to obtain a good RTT_{min} estimations.

Using a TCP LoLa Linux implementation and an emulated network testbed, TCP LoLa authors [75] state this algorithm is able to achieve high link utilisation, low queuing delay and good scalability in 100Mbps and 10Gbps links. However, the main weakness of this algorithm is it cannot coexist fairly with loss-based CC in typical FIFO queue management. It relies completely on the bottleneck (e.g. using AQM or isolating loss-based flows from low-latency flows in separate queues) to provide fair share when competing with loss-based flows. Therefore, it is practically very hard to deploy this CC protocol globally. Additionally, it is not clear how this algorithm behaves in shallow buffers and how it reacts to packet loss.

B. Dual Mode Approaches

Since delay-based CC algorithms respond to congestion feedback much earlier than loss-based CC, delay-based flows realise low link capacity sharing when competing with loss-based flows especially when bottleneck's buffer is large. Dual mode CC approaches work around that issue by switching to aggressive mode (loss mode) as soon as buffer filler flows are detected. They stay in the loss mode for an interval or until loss-based flows finish, then they return to normal delay mode. Different algorithms have different loss-based flows detection techniques but all of them use the delay signal in that matter. Table II summarises the dual mode TCP variants reviewed in this section.

1) TCP Vegas+: Hasegawa et al. [77] proposed TCP Vegas+ to address some of the fairness issues identified with TCP Vegas (section V-A2). This algorithm borrows the aggressive *cwnd* growth function of TCP Reno and the moderate TCP Vegas approach to produce a hybrid congestion control technique. In CA, TCP Vegas+ employs TCP Vegas algorithm when no loss-based flow is detected, and moves to TCP Reno *cwnd* increase mode if an aggressive flow is inferred to be sharing the bottleneck.

TCP Vegas+ uses the following heuristic to detect the loss-based flows based on the trend of RTT. One every received ACK, a state variable *count* is incremented by 1 if Vegas+ detects an increase in the current RTT while *cwnd* is not increased. On the contrary, it decrements *count* by 1 when RTT decreases while *cwnd* is not increased. Moreover, the algorithm halves *count* when packet loss is detected using the 3DUPACK mechanism (see Section IV-A1) and resets it when the loss is detected using the RTO timer.

If *count* reached a predefined threshold (such as 8), the algorithm moves to the aggressive (loss-based) mode, and it returns to the moderate (delay-based) mode when *count* becomes zero.

The notion of the loss-based flow detection algorithm is that in a stable network, RTT should not increase when cwnd is unchanged unless there is a Reno-like flow competing for the bottleneck bandwidth. If the algorithm sees such RTT increase, it assumes another loss-based flow is sharing the bottleneck so must itself moves into the loss-based mode. On the other hand, the algorithm moves back to Vegas mode as soon as a packet loss is detected because that loss could happen due to the aggressive *cwnd* growth of TCP Vegas+ flow itself. Vegas+ uses the count threshold as an attempt to reduce the false positive detection of loss-based flows.

Although this approach attempts to solve the friendliness problem, it does not address the other issues of TCP Vegas such as rerouting problem. Additionally, in some environments that include high congestion or high RTT fluctuations (such as wireless networks), TCP Vegas+ could enter the aggressive mode and never exit from it due to wrong RTT measurements.

TABLE II
DUAL MODE CC APPROACHES REVIEWED IN SECTION V-B

TCP variant	Section	Algorithm aims	Delay signal type	Metrics
TCP Vegas+ [77]	V-B1	solves TCP Vegas inter-protocol fairness issue	RTT	queue occupancy
YeAH TCP [93]	V-B2	scalability, low stressing on the network, low queuing delay, non-congestion related tolerance	RTT	queuing delay, queue occupancy
CDG [10]	V-B3	low queuing delay, tolerance to non-congestion related losses, standard TCP compatibility	RTT	RTT_{min} and RTT_{max} gradient
Cx-TCP [94]	V-B4	low queuing delay, coexistence with loss-based flows fairly	RTT	queuing delay
Copa [95]	V-B5	high throughput, low queuing delay, coexistence with loss-based flows fairly	RTT	queuing delay
Nimbus [96]	V-B6	high throughput, low queuing delay, coexistence with loss-based flows fairly	RTT,	queuing delay, RTT_{min}

This makes the algorithm act as loss-based most of the time, eliminating the advantages of the delay component.

Moreover, TCP Vegas+ evaluated by the authors [77] using simulated network only in which RTT measurements are not realistic since it does not reflect the noise of RTT signal in the real world or emulated environments.

2) YeAH TCP: Yet Another Highspeed TCP (YeAH) is a hybrid congestion control algorithm by Baiocchi et al. [93] that aims to achieve high throughput in large BDP networks but without stressing the network. YeAH originates from the observation that other high speed CC algorithms (such as HS-TCP [97] and STCP [98]) improve throughput in large BDP networks at the cost of high ‘stress’ on the network, causing frequent congestion events with large number of packet losses as well as high queuing delay.

Similar to TCP-Africa (Section V-C1), YeAH works on one of two modes at a time depending on the congestion level. In fast mode, the congestion window increases aggressively using STCP rules while in slow mode TCP Reno rules are applied. The decision of changing from one mode to another is also based on Vegas-like estimation of the number of packets in the bottleneck buffer and congestion level estimation (TCP DUAL-like metric). However, these estimations are redefined by YeAH in such way that RTT_{base} is the minimum RTT seen during the connection lifetime, RTT sample ($RTTmin_i$) is the minimum RTT seen during the transmission of last window (i.e. measured once per RTT) and the congestion level is calculated as a proportion to the RTT_{base} but not to the Q_{max}. Formally, it calculates the queue delay (Q_i) using equation 13, queue size (Δ_i) using equation 14 and the congestion level (L_i) using equation 15

$$Q_i = RTTmin_i - RTT_{base} \tag{13}$$

$$\Delta_i = Q_i . \left(\frac{cwnd_i}{RTTmin_i} \right) \tag{14}$$

$$L_i = \frac{Q_i}{RTT_{base}} \tag{15}$$

If $\Delta_i < \delta$ and $L < 1/\varphi$, the algorithm switches to the fast mode, otherwise the slow mode is used. δ is a tunable constant (for example, 80 packets) which governs the number of packets pushed by one flow in the bottleneck buffer.φ is another tunable constant (for example, 8) the limits the congestion level caused by all flows sharing a bottleneck.

Moreover, a precautionary de-congestion algorithm is utilised in the slow mode to control the queuing delay and buffer overflow. Whenever $\Delta_i > \delta$ and with no Reno-like greedy flows competing for the bottleneck, *cwnd* is reduced by Δ_i and *ssthersh* is set to $cwnd/2$ once per RTT. YeAH detects the competing greedy flows based on the mode switching behaviour of the algorithm in order to achieve inter-protocol fairness. The algorithm calculates $count_{fast}$ which is the number of RTTs the algorithm spend in the fast mode and $cwnd_{reno}$ representing an estimation for the congestion window of the greedy flows (maintained using Reno rules). If $count_{fast}$ becomes greater than a threshold, $cwnd_{reno}$ is set to $cwnd/2$ and $count_{fast}$ is reset as an indication for competing with other non-greedy flows.

The precautionary de-congestion can be applied only if the algorithm is in the slow mode and $cwnd > cwnd_{reno}$, otherwise Reno-style window growth is used.

Finally, based on TCP Westwood (Section V-D1), the algorithm exploits the queue size after packet loss to find an optimum window size when the loss is not related to network congestion. This can improve the algorithm performance in lossy environments such as wireless networks.

Experimental evaluation shows that YeAH is able to realise very good throughput and low queuing delay in fast and long distance network as well maintaining intra- and RTT-fairness and friendliness. However, this approach needs more evaluation in more complex networks and scenarios to confirm the robustness against delay signal noise and distortion.

3) CAIA-Delay Gradient (CDG): CAIA-Delay Gradient (CDG) [10] is a hybrid CC algorithm that tries to maintain low queue delay and reasonable fairness by using delay-gradient CC when possible, and loss-based CC when competing with loss-based CC algorithms. CDG is also able to distinguish between congestion related and random loss, behaving differently to achieve high goodput in lossy environments such as a wireless network.

CDG uses delay-gradient measurements to detect network congestion and bottleneck queue states (full, empty, rising and falling). The notion of estimating the queue state allows CDG to differentiate between congestion and random losses. CDG considers the loss is congestion related only if the queue state is full and never backs off *cwnd* when the losses are not congestion related.

As instantaneous RTT measurements are noisy, CDG calculates the average smoothed delay-gradient ($\bar{g_n}$) using RTT_{min} and RTT_{max} seen in an RTT measured interval. The smoothed average further reduces noise. The current CDG implementation uses $\bar{g_n}$ calculated from RTT_{min} or RTT_{max} to estimate congestion.

When no congestion occurs, CDG increases *cwnd* by one MSS per RTT.

When congestion is detected using the loss signal and the queue state is full, CDG halves *cwnd*. If the queue state is not full, *cwnd* is unchanged.

When congestion is detected using the delay gradient signal, equation 16 is used to probabilistically determine if CDG should back off *cwnd*. The exponential factor achieves fairness between flows with different base RTTs, while G is a scaling factor that determines the algorithm aggressiveness.

$$p_{backoff} = 1 - e^{-\left(\frac{\bar{g_n}}{G}\right)} \tag{16}$$

When probabilistic backoff occurs, CDG decreases *cwnd* by a backoff factor $\beta = 0.7$. The higher factor for β allows CDG to maintain high link utilisation.

To help compete with loss-based flows, CDG uses the *loss-based shadow window* technique first described in [99] and *ineffectual backoff* mechanism. The shadow window mimics TCP Reno window growth. CDG sets *cwnd* to the shadow window as soon as a packet loss is detected. On the other hand, ineffectual backoff is used to detect competing loss-based flows so CDG can move to loss-based mode for a certain interval.

The authors of CDG [10] claim that at 1% non-congestion related packet loss, CDG achieves 65% bandwidth utilisation compared with TCP NewReno at 35% under the same network conditions. At the same time, CDG keeps bottleneck queues short (particularly compared to loss-based CC).

Despite trying to compete with loss-based CC, early back-off by CDG results in it being unable to attain fair capacity sharing with loss-based CC. For this reason, and because of its low latency, Armitage et al. [5] propose using CDG as an LPCC for home networks to reduce the impact of background traffic on latency-sensitive applications. Tangenes et al. [100] evaluated CDG and also concluded that it is a good candidate to be used as a deadline-aware LPCC as its priority can be dynamically adapted using the scaling parameter G from equation 16.

4) Cx-TCP: Coexistent TCP (Cx-TCP) [94] is another loss-delay hybrid congestion control that attempts to provide low latency transport while achieving better coexistence with loss-based flows. Budzisz et al. [94] were inspired by the Probabilistic Early Response TCP (PERT) [73] algorithm to use a probabilistic back-off function based on delay measurements at the end host.

The main difference between PERT and Cx-TCP is the back-off probability function behaviour. PERT back-off probability function increases when the delay exceeds a threshold (th_{min}) until it becomes one when the delay exceeds another threshold ($2.th_{max}$). On the other hand, Cx-TCP back-off probability function increases until the queuing delay exceeds a specific threshold (Q_{th}). After that point, the back-off probability decreases as the queueing delay increases. When the queuing delay reaches the maximum value (Q_{max}), the probability becomes a very small value (i.e. protocol becomes a full loss-based). This function allows Cx-TCP to coexist with loss-based flows more fairly.

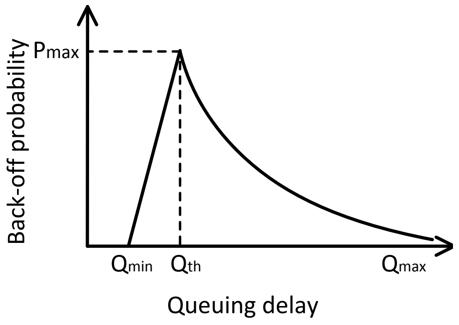


Fig. 13. Cx-TCP back-off probability function

The rationale of the Cx-TCP back-off probability function is that competing delay-based flows do not introduce a large queuing delay, so queueing delay should generally be lower than Q_{th}. However, competing loss-based flows will cause the queueing delay to exceed Q_{th}. Therefore, Cx-TCP should reduce the number of back-off events to better coexist with loss-based flows. When the loss-based flows leave the bottleneck, Cx-TCP reverts to its low latency mode as the queuing delay decreases below Q_{th}.

Using analytical model and simulation, Budzisz et al. [94] show that Cx-TCP is able to achieve better coexistence when competing with loss-based flows while maintaining low queuing delay in the absence of loss-based flows. However, this algorithm assumes that the sender is able to obtain accurate queuing delay measurements which is hard to achieve in many realistic scenarios. Additionally, Cx-TCP flows may obtain a low bandwidth share if the bottleneck has a shallow buffer that does not allow queueing delay above the Q_{th} threshold.

5) Copa: Arun et al. proposes a new loss-delay hybrid congestion control for the Internet called Copa [95]. Copa aims to achieve high throughput and low queuing delay and to coexist with loss-based flows fairly.

The authors of this algorithm state that the bottleneck bandwidth can be estimated as the inverse of the queuing delay. Therefore, they define target sending rate th_{target} to be sending rate at which the sender can transmit to achieve full bandwidth utilisation and low latency.

th_{target} is calculated as $th_{target} = 1/(\delta.Q_i)$ where δ is an adaptive parameter that controls the tread-off between throughput and queuing delay, and Q_i is the estimated queuing delay calculated similar to DUAL (V-A1) but using $RTT_{standing}$ instead of current RTT. $RTT_{standing}$ is RTT_{min} measured in the previous $RTT/2$ interval to remedy ACK compression and signal noise. Copa also calculates the actual sending rate th is calculated similar to Vegas (V-A2) but also using $RTT_{standing}$ instead of current RTT.

On each receiving ACK, If $th_{target} > th$, *cwnd* increases by $v/(\delta.cwnd)$ otherwise *cwnd* decreases by $v/(\delta.cwnd)$. v (defaults to 1) controls *cwnd* increase/decrease speed and its value is changed based on the direction of *cwnd* trend to make Copa flows to converge quickly to full bandwidth utilisation.

This algorithm works in another mode, called competitive mode, to be able to coexist with loss-based flows. It first detects competing loss-based flows and then adjusts δ dynamically to make Copa's flows as aggressive as loss-based flows (i.e. behaving similarly to loss-based scheme). Detecting loss-based flows is based on Copa working behaviour which involves draining the queue periodicity. Thus, if the queuing delay does not reach 10% of maximum queuing delay measured in last four RTTs intervals, buffer-filling flows are assumed to be competing with a Copa's flow. Otherwise, the algorithm works in the default mode.

Copa also changes SS exit condition to be $th_{target} < th$ which provides fast convergence with low latency. Moreover, Copa uses packet pacing to reduces traffic burntness.

Through simulation and user-space implementations, Arun et al. [95] claim that this algorithm is able to achieve similar throughput as TCP CUBIC but with much lower latency and better RTT-fairness. They also state that Copa coexists with CUBIC fairer than TCP BBR (Section V-D5). However, it is not clear how well this algorithm performs in links with very unstable latency such as wireless networks.

6) Nimbus: Nimbus [96] is a rate-based loss-delay dual mode congestion control algorithm that aims to achieve low queuing delay and high throughput while fairly coexisting with loss-based flows.

Nimbus maintains a threshold-based, positive queueing delay to both ensure full link utilisation and to estimate the cross traffic rate. Nimbus calculates a sending rate equal to the bottleneck rate minus the cross traffic rate. Formally the sending rate is calculated using Equations 17 and 18.

$$D(i) = \beta \frac{C}{RTT_i}(RTT_{min} + Q_i - RTT_i) \tag{17}$$

$$S(i+1) = (1-\alpha)S(i) + \alpha(C - z(i)) + D(i) \tag{18}$$

Where α=0.8, β=0.5, C is the bottleneck capacity and z is the cross traffic rate estimation. RTT_{min} is the minimum RTT, RTT_i is the current RTT and Q_i is the current queuing delay. Bottleneck capacity *C* can be estimated using any bandwidth estimation technique such as in [101], [102], [103], [104]. Due to ACK compression and other problems, Nimbus implementation uses the maximum received rate as estimation for bottleneck capacity.

Nimbus models the elasticity of cross traffic to infer the existence of competing loss-based flows, and then switches to TCP-competitive mode (CUBIC-like). This algorithm calculates the periodicity behaviour of link capacity using the Fast Fourier transform. Nimbus uses the observation of high frequency behaviour to conclude the presence of competing loss-based flows. When the frequency becomes low, Nimbus switches back to delay-based mode.

Using user-space implementation with emulated and real-world experiments, Goyal et al. [96] evaluate Nimbus and show that this algorithm is able to detect competing CUBIC and Reno like flows and achieve fair bandwidth share and low queuing delay (lower than BBR (Section V-D5)). However, the authors of this algorithm state that this algorithm is unable to detect competing BBR flows in shallow bottleneck buffers. Therefore, Nimbus flows obtain lower bandwidth share. Additionally, Nimbus considers competing delay-based flows (such as Vegas) as elastic flows and therefore it obtains much higher capacity sharing after moving to TCP-competitive mode.

C. Dual Signal Approaches

Due to the limitations of using the loss signal, some TCP CC variants introduce the delay signal in their work as a supplementary signal in addition to the loss feedback.Generally speaking, the dual signal CC approaches are designed to be scalable in fast and long-distance network environments without stressing the network by increasing *cwnd* rapidly when the queue is short and moving to slow standard *cwnd* growth after the queue becomes long. In addition to that, they attempt to maintain RTT fairness and compatibility with standard TCP. Table III summarises TCP variants that use both the delay signal (secondary signal) and loss signal reviewed in this section.

1) TCP Africa: King et al. [105] propose a CC algorithm called Adaptive and Fair Rapid Increase Rule for Scalable TCP (Africa) to improve TCP scalability in large BDP networks.

TCP Africa operates in one of two regimes: aggressive *cwnd* growth and conservative Reno-like *cwnd* increase. It switches between the two regimes depending on the estimated network congestion level in order to achieve fast convergence and fairness to standard TCP.

The congestion level is estimated using Vegas-like metric Δ (see Section V-A2). It then compares Δ with a threshold (α) to determine which mode should be used. If $\Delta < \alpha$, the *fast mode* is used in which $cwnd$ increases aggressively uses the HS-TCP [97] CA and fast recovery rules to achieve scalability. Otherwise, TCP Africa moves to the *slow mode* in which Reno-like $cwnd$ growth style is used to achieve fairness i.e. increases the window by one MSS per RTT when no loss is detected and halves it on packet loss.

The value of α is chosen to be small constant greater than 1 (α =1.641 in [105]) and it affects the protocol performance. The authors found that no single α is optimal for all networks and conclude that more study is needed to make the value α auto-tuned.

Simulations with ns2 show that TCP Africa can scale quickly to full link utilisation, adapt quickly to network condition changes, causes low packet loss rate as well as good fairness and friendliness prosperities. However, real world experiments should be conducted to confirm these results.

2) Compound TCP (C-TCP): Compound TCP (C-TCP) [106] is a compound loss-delay-based CC that aims to achieve high throughput in high speed high delay networks. Similar

TABLE III
DUAL SIGNAL TCP VARIANTS REVIEWED IN SECTION V-C

TCP variant	Section	Algorithm aims	Delay signal type	Metrics
TCP Africa [105]	V-C1	scalability in large BDP networks, friendliness	RTT	queue occupancy
C-TCP [106]	V-C2	scalability in large BDP networks, friendliness	RTT	queue occupancy
TCP Libra [107]	V-C3	scalability and maintain the compatibility with the standard TCP, RTT fairness	RTT	queuing delay
TCP-Illinois [108]	V-C4	scalability and fairness	RTT	queuing delay

to TCP-Africa, it relies on the delay signal to increase *cwnd* quickly when no congestion is detected and loss signal to achieve fairness when competing with other flows.

C-TCP maintains two congestion windows, one is a standard window W_{reno} managed by Reno-style mechanism and the second is a scalable delay window W_{fast} based on TCP Vegas like algorithm. The congestion window *cwnd* that is used to control the outstanding data is the summation of W_{reno} and W_{fast}.

When Vegas queue size estimator detects small queue ($\Delta < \alpha$), W_{fast} is increased according to a modified AIMD borrowed from HS-TCP algorithm [97]. On the other hand, when Vegas estimator exceeds the threshold ($\Delta > \alpha$), W_{fast} is gradually decreased by $\zeta \times \Delta$ where ζ is a pre-defined constant ($\zeta = 30$ in [106]). This approach provides fast convergence when the queuing delay is short as well as a smooth transition from the scalable congestion control to the standard TCP style.

Real-world and simulation experiments results [106] show C-CTP exhibiting good goodput and inter/intra-fairness in a large BDP environment. C-CTP is used by default for older versions of Microsoft Windows operation system [109] and replaced by TCP CUBIC in Windows 10 Fall Creator update and Window Server 2016's 1709 update [110]. The main reason for abandoning C-TCP in Windows OS is the sensitivity of the delay component to delay fluctuations which cases low performance in many cases [110].

As the scalable component of C-TCP is basically the TCP Vegas buffer estimator, it suffers similar fairness and latecomer advantage issues when base RTT is wrongly estimated (Section IV-A2).

3) TCP Libra: TCP Libra is a TCP CC proposed by Marfia et al. [107] to remedy the RTT-unfairness problem of NewReno when sharing a bottleneck link, improve scalability and maintain compatibility with standard TCP. TCP Libra utilises the delay signal to control *cwnd* growth/decline speed in order to become RTT independent.

Instead of increasing *cwnd* by one MSS every RTT, Libra increases the congestion window according to the equation 19.

$$cwnd_{i+1} = cwnd_i + \frac{\alpha_i}{cwnd_i} \frac{RTT_i^2}{RTT_i + T_0} \tag{19}$$

On packet loss, Libra decreases the congestion window according to equation 20.

$$cwnd_{i+1} = cwnd_i - \frac{T_1.cwnd_i}{2(RTT_i + T_0)} \tag{20}$$

where T_0 and T_1 are constant parameters (eg. $T_0 = 1$ and $T_1 = 1$). T_0 controls the algorithm's sensitivity to the RTT and T_1 is the multiplicative decrease factor. α is a control function that aims to improve the convergence speed, the scalability and the stability of the protocol, where here k_1 is a protocol constant (eg. 2) and C represents the link capacity in Mbps estimated using the CapProbe technique [111] (Equation 21).

$$\alpha = k_1 \, C \, p \tag{21}$$

p is a penalty factor that controls the the window increase step when the network is congested based on queue delay estimation borrowed from TCP DUAL (Section V-A1). p is defined in Equation 22 where k_2 is a constant (e.g 2) that controls link utilisation and protocol friendliness, Q and Q_{max} are the current and maximum queue delays respectively.

$$p = e^{-k_2 \frac{Q}{Q_{max}}} \tag{22}$$

The estimated capacity C in Equation 21 allows the congestion window to converge rapidly to fully utilise available bandwidth, while p reduces the window growth steps exponentially when the queue delay increases. Moreover, $RTT^2/(RTT_i + T_0)$ control RTT-fairness of the protocol in which as the RTT becomes close to T_0, the $cwnd$ growth speed becomes faster.

The multiplicative decrease function (equation 20) shows that *cwnd* is driven not only by T_0 and T_1 but also by RTT_i. This prevents a large *cwnd* decrease after packet loss, providing better throughput in high RTT paths. However, this approach goes against the accepted logic of CC as a large RTT followed by packet loss is typically inferred as a clear sign of congestion. As such, *cwnd* decrease should be larger to control the congestion.

Using ns-2 simulator, the authors evaluate Libra and compare its performance with other TCP protocols [107], showing that Libra can achieve good fairness and high link utilisation in many scenarios. However, in some scenarios TCP Libra has lower performance comparing with TCP SACK and TCP FAST. The authors conclude the the protocol needs more evaluation using real world networks and in more complex scenarios.

4) TCP-Illinois: TCP-Illinois is a TCP CC proposed by Liu et al. [108] that utilises the loss signal as a primary congestion feedback and the delay signal (queuing delay) as a secondary congestion signal to improve the the scalability and fairness of TCP Reno. The underlying idea is similar to TCP Africa (Section V-C1) but rather than use HS-TCP [97] constants, the increase (α) and decrease (β) factors are set to be functions of the queue delay Q_i (similar to TCP DUAL - Section V-A1).

TCP-Illinois sets the increase factor to α_{max} if $Q_i < Q_1$ and α_{min} if $Q_i > Q_{max}$; otherwise it sets α to a concave function inversely proportional to Q_i between α_{max} and α_{min}. Moreover, it sets the decrease factor to β_{min} if $Q_i < Q_2$ and β_{max} if $Q_i > Q_3$; otherwise it sets β to a linear function directly proportional to Q_i between β_{min} and β_{max}. Q_{min} and Q_{max} are minimum and maximum queue delay seen during the connection lifetime, and Q_1, Q_2 and Q_3 are proportions (e.g. 0.01, 0.1 and 0.8 respectively) of Q_{max}. α_{min}, α_{max}, β_{min}, β_{max} are protocol constants (e.g. 0.3, 10, 0.125 and 0.5 respectively).

This algorithm updates α and β once every RTT but does not allow α to be set to α_{max} unless the queue delay stays below Q_1 for a specific amount of time (for example, 5 RTTs) to mitigate the effects of fluctuation in queue delay measurement which can be caused by noisy RTT measurement and packet bursts. Moreover, to improve the fairness with TCP NewReno, the algorithm moves to compatible mode (Reno's α and β coefficients) whenever the window size is less than a threshold (for example, 20KB).

The authors create a mathematical model to analyse and compare TCP-Illinois with other CC algorithms, and conduct simulation-based experiments to evaluate their algorithms' performance [108]. Their results show that TCP-Illinois achieves higher link utilisation in large BDP networks compared to TCP NewReno as well as maintaining protocol intra- and inter-fairness. Moreover, the mathematical model shows that the proposed protocol causes the competing flows to backoff asynchronously which allows the congestion window of the flows to be similar in size and therefore improve overall link utilisation and fairness. However, as this approach primarily uses the loss signal to detect congestion, it cannot control the queue delay and therefore it can lead to bufferbloat problem in large bottlenecks buffers.

D. Delay-sensitive AlgorithmsV-D

Some congestion control algorithms use the delay metric for specific purposes not directly related to the congestion feedback signal. These techniques use the delay signal to calculate an effective congestion window size after packet loss event and/or differentiate between congestion and non-congestion related packet losses to improve the performance in wireless networks. Others uses the delay metric to calculate reasonable bytes inflight limit to realise both high throughput and low latency. Table IV summarises the delay-sensitive TCP variants reviewed in this section.

1) TCP Westwood/Westwood+: Mascolo et al. [112] proposed TCP Westwood (TCPW) CC algorithm to solve the low performance of NewReno in intrinsically lossy and fast networks. TCPW modifies window multiplicative decrease mechanism during fast recovery to calculate an optimum window size based on the estimated bandwidth (BWE) and RTT_{base}. The goal is for window size at any time to be approximately equal to the path's BDP, and hence achieve full link utilisation with minimal undesirable queuing.

Westwood sets the window size and *ssthresh* to $BWE \times RTT_{min}$ when packet loss is detected, which is usually a less aggressive reduction than simply halving *cwnd*. TCPW uses RTT_{min} as an estimation for RTT_{base}. RTT_{min} is measured as the smallest RTT sample seen during the connection lifetime.

Westwood's strategy for estimating the bandwidth relies on detecting the ACK receiving rate. If the receiver generates an ACK packet directly after receiving a data packet, the rate of ACK packets observed by the sender will be same as the rate of data packets received by the receiver i.e. the receiving rate equals the rate that the bottleneck supports. Then, the estimation of the utilised bandwidth in the forward path can be calculated by multiplying the ACK rate by number of bytes acknowledged by ACK packet. Even when some ACKs are lost or the receiver decides to use delayed ACK mechanism, the calculation of long-term estimation will not be significantly affected. Although the delayed ACK and ACK packet loss reduce the ACK rate, the ACK packet will carry acknowledgement for larger amount of data leading to an acceptable estimation.

TCP Westwood applies two-stage bandwidth estimation procedure to reduce the impact of fluctuation on the measurement. In the first stage, the bandwidth sampleb_k is calculated as $b_k = d_k/\Delta_k$where d_k is the amount of data that has been acknowledged and $\Delta_k$is elapsed time since the receiving of the previous ACK. In the second stage, the final BWE is obtained by applying a low-pass discrete time filter:

$$BWE_k = \alpha.BWE_{k-1} + (1-\alpha)\,\frac{b_k + b_{k-1}}{2}$$

where BWE_k is the filtered bandwidth estimation, BWE_{k-1} is the previous estimation,α is a constant ($\alpha = 0.9$ for example) and b_k, b_{k-1} are the current and the previous bandwidth estimation samples respectively.

The evaluation results [112] show remarkable throughput improvement in the presence of random errors compared with TCP NewReno as well as very good inter- and intra-fairness. However, Grieco et al. [119] discovered that the estimator overestimates the bandwidth and causes unfriendliness in certain conditions. These conditions include ACK-compression effect that clusters ACK packet arrivals due to congestion and queue fluctuation in the reverse path [120] and employing AQM in the router [121].

A slightly modified version of this algorithm, called TCP Westwood+, was proposed to solve the ACK compression effect [113] by computing the bandwidth samples every RTT i.e. $b_k = d_k/RTT_k$ where d_k is the amount of data that has been acknowledged during the last RTT. The final BWE value is obtained by applying exponentially weighted moving average to the bandwidth samples b_k.

With respect to the RTT measurement, TCP Westwood creates a persistent queue in the bottleneck buffer if the RTT_{min} is larger than the actual RTT_{base}, such as in statistical multiplexing backoffs where many flows share the same bottleneck. Wrong RTT_{base} estimation leads to a congestion window larger than BDP after packet loss event causing unfairness between the competing flows.

2) TCP Westwood-Based Algorithms: In this section we briefly summarise a number of modifications proposed to

TABLE IV
DELAY-SENSITIVE TCP VARIANTS REVIEWED IN SECTION

TCP variant	Section	Algorithm aims	Delay signal type	Metrics
TCP Westwood [112]	V-D1	high throughput in lossy environments	RTT	RTT_{min}
TCP Westwood+ [113]	V-D1	high throughput in lossy environments, improved bandwidth estimator	RTT	RTT_{min}
TCPW CRB [11]	V-D2	high throughput in lossy environments, better bandwidth estimator, friendliness	RTT	RTT_{min}
TCPW ABSE [114]	V-D2	high throughput in lossy environments, better bandwidth estimator, friendliness	RTT	RTT_{min}
TCPW BR [115]	V-D2	high throughput on heavy non-congestion related losses, friendliness	RTT	RTT_{min}
TCP-AP [116]	V-D3	high throughput in multihop wireless networks	RTT	coefficient of RTT variation
RAPID [117]	V-D4	Full bandwidth utilisation in fast network with dynamic bandwidth, low queuing delay and fairness	OWD	indirect queuing delay
TCP BBR[118]	V-D5	low queuing delay, high throughput	RTT	RTT_{min}

remedy the vulnerabilities of TCP Westwood's bandwidth estimation technique.

TCP Westwood Combined Rate and Bandwidth estimation (TCPW CRB) [11] aims to improve the efficiency and friendliness of TCPW. It uses Rate Estimation (RE), which is long-term bandwidth estimation similar to what is used in Westwood+ but measured over a constant time interval (T) instead of RTT, in addition to Bandwidth Estimation (BE) of TCPW. The rationale of using two estimators is that RE prevent overestimation of the bandwidth when the network is suffering from congestion, but it underestimates the bandwidth when non-congestion related packet losses occur. TCPW CRB calculates the congestion window size as $BE \times RTT_{min}$ when packet loss is caused by random transmission error, and $RE \times RTT_{min}$ when the loss is caused by network congestion. TCPW CRB assumes that the loss is congestion related if $cwnd/(RE \times RTT_{min})$ upon packet loss is is smaller than a threshold (for example, 1.4); otherwise, the loss is assumed to be a random loss.

The authors of TCPW CRB claim that the dual bandwidth estimation method improves the trading-off of TCPW efficiency and friendless to NewReno [11]. However, their claim has been confirmed only using ns-2 simulation experiments. Additionally, they conclude that more study is needed to understand the impact of different conditions, such buffers sizes, the error rate and AQM, on the friendliness of the algorithm.

TCPW CRB authors later found that the time interval of RE should not be constant during connection lifetime to provide better friendless. Therefore, they proposed TCPW Adaptive Bandwidth Share Estimation (TCPW ABSE) [114]. TCPW ABSE adaptively changes the CRB sampling intervals depending on a congestion level estimation heuristic. The network congestion level is estimated based on the difference between the averaged sending rate sample ($VE = cwnd/RTT_{min}$, Vegas estimation) and throughput sample (RE, Westwood+ estimation). If the difference is large, the network is considered congested and large interval (one RTT) is used; otherwise a small interval is used. The authors state that this technique produces more precise bandwidth estimation in different congestion levels. Although ns-2 simulation shows that TCPW ABSE is able to achieve very good fairness with TCP NewReno as well as fast response to network conditions changes, real world experiments are required to validate the results.

Yang et al. [115] showed that the previous TCPW-based enhanced algorithms improve the throughput significantly in lossy environments but with random error rate < 2%. Therefore, they propose TCPW with bulk repeat (TCPW BR) to improve the performance in extreme loss conditions. TCPW BR uses dual mechanisms to discriminate between congestion and non-congestion losses: the queue delay threshold method based on TCP DUAL (section V-A1) and comparing bandwidth estimation (BE) to the expected throughput (VE) similar to TCPW ABSE. If the packets losses seem to be caused by channel transfer error, TCPW BR resends all packets in flight, freezes RTO value and leaves *cwnd* unchanged; otherwise it reacts to the losses same as NewReno. Using ns-2 simulation, the author confirm the efficiency of the algorithm even at high error rate (>5%) and the friendliness to TCP NewReno.

There are more TCPW variant techniques including: TCPW Bottleneck Bandwidth Estimation (BBE) [121] which aims to solve TCP Westood unfriendliness problem in different network conditions including highly varying bottleneck buffer sizes, AQM employment and heterogeneous flows RTT.

TCP Westwood with agile probing (TCPW-A) [122] aims to provide better performance in highly dynamic bandwidth networks as well as lossy environments. It achieves that by repeatedly resetting $ssthresh$ based on BE like estimation and increasing $cwnd$ exponentially (when $ssthresh$ is lower than the estimation) in SS phase to quickly converge to full bandwidth utilisation, and does the same thing in the CA mode if a persistence non-congestion is detected; otherwise it uses NewReno $cwnd$ increase function.

Similar to TCPW, all these modified algorithms can suffer from unfairness problems in different degrees if the RTT_{min} is an overestimation of RTT_{base}. Additionally, it is not clear how they react to the bottlenecks that utilise AQM to control the congestion in their buffers.

3) TCP-AP: TCP with Adaptive Pacing [116] is specialist hybrid window/rate-based CC that proposed to improve low performance of IEEE 802.11 multihop wireless networks. More specifically, this approach aims to reduces reduces link layer contention by adjusting packet sending rate based on contention estimation of the path. While window (*cwnd*) size controls the number of bytes in-flight, the pacing rate is used to provide smooth packet sending and preventing packets bursts.

This CC does not change any of standard slow-start, congestion avoidance and congestion recovery algorithms. It only controls the pacing rate of packet sending when *cwnd* allows transmitting new data.

TCP-AP mainly uses two metrics to calculate packet pacing rate. The first metric is the coefficient of RTT samples variation which is used to estimate contention degree of the path. The second metric is called out-of-interference delay which is the time between sending a packet from a node and receiving that packet at a second node existing outside the signal collision range of the first node.

The authors of this algorithm [116] claim that TCP-AP is able to achieve 10 times higher throughput than TCP NewReno in an emulated wireless network with 20 nodes. They also state that this algorithm provides good fairness with other flows and responses quickly to network condition changes. However, this algorithm relies on the RTT measurement which suffers from ACK compression and other packet delays caused by wireless link layer reliability (see Section VI). Moreover, this algorithm requires additional information from the operating systems, such as number of hops and data link parameters, which makes the kernel-space implementation complicated.

4) RAPID: RAPID congestion control [117] is rate-based approach that aims to realise fairness and high throughput in fast networks with dynamic bandwidth while maintaining inter- and intra-protocol fairness and low queuing delay. Although this algorithm does not utilise the delay signal explicitly in its operation, the theory behind RAPID is based on the queue delay growing up.

The authors of this algorithm claim that conventional TCP CC algorithms are not able to achieve full bandwidth utilisation in fast networks with variable bandwidth due to slow bandwidth probing technique of those algorithms. More specifically, conventional TCP bandwidth probing requires one RTT time interval to realise the result of a new probing cycle and the new rate probing is not much larger than the previous rate (*cwnd* typically increases by one MSS every RTT interval). Therefore, many RTT intervals are required before converging to full bandwidth utilisation.

RAPID is able to probe multiple rates by sending groups of N packets with N-1 different rates in each group i.e. increases the gap between the sending packets of a group. At the receiver host, the inter-packet arrival times (the gaps) of each group is monitored. If the gaps trend increases (i.e. $gap_i > gap_{i-1}$), this means the bottleneck queue started to build-up since every packet sent at a faster speed then its preceding will cause additional delay in packet delivery (larger gap). Then the estimated bandwidth will be the rate before seeing positive gaps trend (i.e. $rate_{i-1}$). If $gap_i \leq gap_{i-1}$ for all groups, the sender generates and transmits more groups with higher sending rate until the receiver obverses positive gap trend (i.e. $gap_i > gap_{i-1}$). The receiver then explicitly sends the value of $rate_{i-1}$ to the sender host to use it as sending rate.

Using ns-2 [84] implementation of RAPID, Konda et al. [117] claim that this algorithm can converge to full bandwidth utilisation of gigabit networks in 1-4 RTTs while keeping the queue short. They also state that RAPID provides good fairness and a small impact on conventional TCP flows. However, this algorithm requires receiver-side modification which makes the global deployment very hard. Additional, RAPID requires high timer resolution to send packets at an accurate rate which is hard to achieve for many hosts and produces additional overhead. Moreover, similar to many delay-based algorithms, RAPID can suffer and produce unpredictable behaviour in wireless networks due to collision avoidance and data link frame recovery of these networks which change packets transmission pattern.

5) TCP BBR: Bottleneck Bandwidth and Round-trip propagation time [118] is a recent congestion control algorithm that aims to solve the bufferbloat problem and improve TCP throughput. Rather than seek a feedback signal to detect congestion, BBR controls congestion by pacing the sending rate according to the currently estimated bottleneck bandwidth BW and RTT_{min}.

A delivery rate sample is calculated as the ratio between delivered data and time elapsed for delivering that data. BBR uses a windowed maximum filter over a 6-10 RTT period for delivery rate samples to obtain a BW estimation.

As a secondary control mechanism, BBR utilises a TCP-like *cwnd* mechanism to limit the maximum amount of inflight data. This window is set to a few multiples of BDP (with BDP calculated as $BW \times RTT_{min}$) to remedy common receiver and network issues such as delayed and aggregate ACKs.

A windowed minimum filter for RTT_i is used to estimate RTT_{min}. If RTT_{min} does not change and no RTT sample matches RTT_{min} for 10 second period, the number of packets inflight is reduced to 4 for a short period to drain the buffer and probe for a new RTT_{min}. This allows BBR to refresh its RTT_{min} estimate as well as permitting flows to converge to a fair share of the bandwidth.

BBR maintains one BDP worth of packets inflight most of the time to guarantee full link utilisation with low queuing delay. Periodically, BBR increases the sending rate and inflight size to 125% for an RTT interval to probe available bandwidth and changes the paced rate accordingly. If BW remains unchanged, the sending rate and inflight size is reduced to 75% to drain the built-up queue. BBR then returns to normal operation of using 100% of the estimated BW and inflight size.

TCP BBR has been implemented in Linux and deployed in Google services since 2015, achieving 2 to 25 times greater throughput than CUBIC [118].

BBR's authors evaluate the algorithm in large variety of scenarios and network environments. They claim that BBR improves throughput and reduces latency compared with TCP CUBIC, while being able to achieve acceptable fairness. However, the authors agree that BBR has problems in specific situations and requires more research to remedy these issues.

For example, BBR flows obtain a lower share of bandwidth when competing with loss-based flows through a bottleneck with a large (several BDPs) buffer. Additionally, BBR flows can suffer from unexpected packet loss when token-bucket traffic policers are used. More specifically, when BBR sends packets faster than the bucket fill rate, all packets are discarded by the policer.

Moreover, Hock et al. [123] conduct experimental evaluation of TCP BBR in an emulated environment and conclude that BBR is able to achieve its goals. However, they also find that BBR produces a large number of packets losses, unfairness and long queuing delay in some scenarios, specially in shallow bottleneck buffer. There is also a big concern in transport research communities regarding BBR packets losses ingratiation as well as ECN support ingratiation.

Although the algorithm attempts to remedy the wrong estimation of RTT_{base} using RTT_{min} measurement reset mechanisms, it is not clear how well this method works in environments with long standing queues causing by other CC algorithms/flows. Therefore, more evaluation is need to understand how the wrong estimation can affect its operation.

VI. Challenges facing the delay signal

A. *Inaccurate delay measurements*

As we mentioned in Section IV-A2, many problems facing the use of delay signal causing undesirable side effects for congestion control techniques that benefit form this signal. Usually, inaccurate delay measurement happens due to queue fluctuation, burst packet sending, delayed ACK, hardware transmission offloading, link layer buffering (especially in wireless networks) as well as inaccurate sampling. Additionally, interference between competing flows causes the delay-based techniques to wrongly estimate the congestion level. This includes wrong propagation delay estimation (which is used by many algorithm) due to route change and sequence of flow starting time (latecomer advantage problem).

Reverse path congestion is another problem that can distort the delay signal when the RTT metric is used. When congestion happens in the reverse path (ACK path), the RTT metric will not reflect the real congestion state of the network (forward path) since the ACK packets are small and do not contribute to the congestion. Therefore, some algorithms utilise the one-way delay signal instead. However, unlike using RTT, the one-way delay metric needs receiver side modification or enabling TCP timestamp option which makes using that signal not a universal solution for standard TCP. Furthermore, the one-way delay signal cannot be used as an absolute delay time without clock synchronisation between source and destination. Moreover, finding universal optimum threshold values for the threshold based algorithms is very hard or unachievable due to large varieties of network environments and conditions. More serious problem for the delay-based algorithm is the unfairness when competing against the greedy loss-based algorithms including the standard TCP congestion control.

B. *Data link layer reliability and channel access method*

Reliability mechanisms and channel access methods in some networks, such as wireless networks, have a significant negative impact on the delay signal usability. The data link layer of modern wireless networks, for example, attempts to supply the network layer with error-free packets by providing reliable packet recovery mechanisms. If packets arrive corrupted or do not arrive at the destination, the sending station will retransmit the packets until they arrive correctly. This behaviour makes the network layer to wait for an unknown interval of time before receiving the packets. The duration of this interval depends on the degree of the noise and interference on the channel.

Moreover, in shared transmission medium, Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) manages channel access and attempts to minimise collisions of transmitted packets. This involves sensing the carrier to see whether it is busy andwaiting for a random intervals until the channel becomes available.

CSMA/CA can also use Request to Send/Clear to Send (RTS/CTS) to reduce the problem of collisions due to hidden nodes. With RTS/CTS, the sender first sends an RTS frame to be received by the Access point (AP). The AP sends back a CTS frame to the sender to indicate that it may begin trasmission. If the channel is busy or the AP has data to send (typically AP has higher priority) the sender should further wait before reattempting the RTS/CTS protocol. If RTS/CTS frames are dropped, the node should wait for a random time period before retransmitting the RTS frame.

This introduced waiting time becomes part of the end-to-end delay measurement used by delay-based CC. This latency is not related to network congestion and it corrupts the delay feedback. Managing medium access becomes more challenging in duplex communication channels. In a point-to-point simplex communication channel, however, the issue becomes less problematic since the channel will be dedicated for one transmitter only.

Both data link reliability and CSMA/CA significantly weaken the correlation between congestion and the measured delay signal. Any congestion related decisions (e.g. *cwnd* backing-off) made by delay-based CC depending on such signal significantly reduces the throughput and produces unpredictable behaviour. In addition, these mechanisms can affect packets on both the forward path and the reverse path causing packets to be sent in bursts. Sending packets in bursts causes queue fluctuation and ACK compression which have a significant negative impact on RTT measurements.

Another problem that causes inaccurate RTT measurements is packet aggregation of some data link layers of wireless networks (e.g. IEEE802.11n). Packet aggregation aims to reduce the overhead of sending small packets by combining multiple packets and transmitting them as one frame. Karlsson et al. [124] have shown that packet aggregation can make several ACK packets to be sent back-to-back causing ACK compression effect.

It is worth noting that even rate-estimation CC techniques (e.g. TCP BBR in Section V-D5) can suffer (but in a smaller degree) from these data link mechanisms since the ACKs can arrive at the sender in burst faster than how the receiver sent. Therefore, the bandwidth estimator will overestimate the available bandwidth causing the sender hosts to transmits packets faster than the actual available bandwidth.

Therefore, it is very difficult and unreliable to use the delay signal to control congestion in noisy and shared medium networks with strong data link reliability. However, using some delay based estimations (e.g RTT_{base}) to improve protocol performance in such networks is considered useful and usable.

C. Emerging communication networks

With wide deployment of the TCP protocol in many devices and operating systems, many emerging network applications utilise this protocol to provide reliable data transfer. Different types of networks have been developed to fulfil the needs of modern network applications. These include vehicular networks (VANETs) [125], mobile ad hoc networks (MANETs) [15] and delay tolerant networks (DTNs) [126]. These types of networks typically have different characteristics to traditional wired networks.

These characteristics can include one or more of changing network paths (due to node mobility), changing numbers of connected nodes, shared multi-hop channels, varying intrinsic delays, unpredicatable medium and data link reliability, high bit error rates, lack of continuous network connectivity and low powered (electricity and/or computation) resources. These challenges can propagate up the stack where they can inpact on the performance of the transport protocols used within these applications.

For example, varying latency due to changing paths or packet loss due to unstable network topologies can result in the TCP RTO being falsely triggered. Similarly, non-congestion related packet losses can occur more frequently due to higher bit error rates in such networks [15].

Further impact may be seen where variable data transmission and unpredictable latencies can cause delay-based congestion control to misinterpret the delay signal. It is difficult to determine if any increase in measured delay is the result of network congestion or other non-congestion related effects.

Another challenging application space is the Internet of Things (IoT). Many end devices (things) utilise a TCP transport to communicate with a number of pre-existing Internet services and devices [120]. Typically, a large number of these devices would connect via wireless networks (infrastructure, Ad hoc or mesh networks).

IoT devices typically have limited processing power and/or energy resources, limiting their capabilities. CC for these devices should be lightweight and efficient to provide high performance with limited resources.

Traditional TCP algorithms may perform poorly under these circumstances. Packet loss results in wasted communication capacity, and can lead to increased processing overhead when recovering from loss. Alternatively, delay-based CC may result in improved overall performance at the increased cost of accurately predicting network state.

Each of these emerging applications have unique network and application requirements, meaning no one CC algorithm is best for all cases. This implies that selection of suitable CC mechanisms should be performed on a case-by-case basis.

D. Effects of using AQM on delay signal

While AQM and delay-based CC share the same goal of keeping queuing delay low, they achieve that goal from different places. AQM's place is the middleboxes (such as switches, routers and firewalls) and they need a reaction from the end host to control the congestion. On the other hand, delay-based CC achieves that goal using end-to-end approach without help from network appliances.

In controlled environments (data centre networks, for example), it is easy to deploy AQM in the bottlenecks since network equipment usually belongs to the one organisation. Additionally, delay-based CC algorithms can work very well in such environments since flows can be homogeneous (one CC is used) and the noise in delay signal is low. Therefore, either of the two approaches works well. On the Internet, however, there is no one organisation having control over the equipment along the path between the sender and receiver hosts. If the place of a bottleneck is known and accessible by an individual, queuing delay can be controlled by deploying AQM in that bottleneck. An example of such bottlenecks is ADSL home gateways in the upstream direction.

On the other hand,bottleneck location is unknown, inaccessible or AQM is not implemented for specific hardware, there is no way for the end-users or service providers (such as gaming servers) to use AQM. Furthermore, it is very hard to know whether AQM is deployed along the path or not. Therefore, in such common case delay-based CC (low-latency transport) is the only choice for the end-users or service providers to achieve low latency communication.

In this context, it is not uncommon to see delay-based CC flows pass across AQM-enabled bottlenecks. This raises questions about the effects of deploying AQM on delay-based flows and the coexistence of AQM and delay-based CC.

AQMs effectively create short queues (in size or time) to prevent packets from unnecessarily residing in the buffer for too long. On the other hand, delay-based CC relies on queuing delay measurement to infer congestion. Short queues reduce the signal variations that latency-sensitive CC algorithms rely on.

Most delay-based and hybrid congestion control algorithms compare the delay measurements with thresholds to infer the congestion level in a network. If these thresholds allow the bottleneck queue to include packets longer than AQM allows, the algorithm will fall back to loss signal reactive mode. This behaviour damages a substantial goal of the delay-based algorithms in which they try to reduce the number of packets losses by early reacting to the congestion.

Many delay-based CC algorithms (TCP DUAL and TCP Vegas for example) aim to reduce TCP saw-teeth *cwnd* behaviour to achieve both stability and high throughput even in shallow bottleneck buffers. However, falling back to loss mode destroys these aims and can create fluctuation in throughput because CC may excessively back off *cwnd* below path's BDP especially in high-speed long-distance links.

Additionally, working in loss-mode all the time can alter the design goal of some delay-based CC. For example, it has been shown that AQM bottlenecks raise the priority of LPCC making scavenger-class transport to compete equally with conventional TCP for bottleneck bandwidth [127]. Without taking AQM deployment into consideration, scavenger-class transports can impact negatively on conventional TCP traffic. A study on the effects of modern AQM bottlenecks on *libutp* [89], LEDBAT-based widely deployed transport, shows that LEDBAT flows become more aggressive than conventional TCP flows when AQM is used [91]. The issue is that *libutp* increases *cwnd* quickly (faster than standard TCP) when the queuing delay is small and *cwnd* growth becomes slower as the queuing delay becomes closer to LEDBAT's target delay (see Section V-A7 for details about LEDBAT CC). This strategy works well with FIFO bottleneck to converge quickly to full utilisation but not with AQM enabled bottlenecks since queuing delay will hardly reach LEDBAT's target delay.

Moreover, if the AQM allows very low queuing delay, the noise reduction techniques used by delay-based algorithms (such as moving average) can destroy the signal completely. Figure 14 illustrates how the use of AQM in a bottleneck can destroy the delay signal through reducing the queueing delay variation.

Despite the challenges of using a delay signal in such environments, AQM can help some algorithms to overcome the latecomer advantage problem by forcing all flows to back off. This allows resistant queues to drain giving an opportunity for flows to obtain correct propagation delay estimation. Additionally, if a bottleneck employs AQM with ECN marking support and the end hosts also support ECN, delay-based CC can work reasonably well in keeping queuing delay low without packet losses since the congestion signal is sent directly from the congested box. However, reacting to ECN signal by the delay-based CC can create fluctuations in throughput similar to packet drop. In this aspect, TCP Alternative Backoff with ECN (ABE) [59] plays a very important role in keeping unacknowledged data close to the path's BDP and reducing *cwnd* fluctuation. This proposal suggests using two different *cwnd* back-off factors; one for ECN (β_{ECN}) and another for packet drop (β_{loss}) and the proposal recommends using 0.8 for β_{ECN}.

Recently, there has been growing interest in deploying modern AQM on the Internet due to increased sensitivity of many applications to latency caused by bufferbloat. This raises questions about how AQM bottlenecks will affect CC algorithms that rely entirely or in-part on the delay signal. Therefore, research needs to be conducted to understand the behaviours of delay-signal CC with AQM in a wide range of environments. This requires more theoretical and practical studies for the delay signal and the distortion level of AQM on that signal and the CC algorithms that employ such a signal.

It is apparent that the presence of an AQM managed bottleneck may damage the delay signal with respect to delay-based CC algorithm, which could result in some implementations reverting to loss-based mode leading to unintended consequences when competing with other flows. However, the short queues enforced by the AQM will limit this impact on the performance of competing flows. Alternatively, the use of delay-based CC algorithms may help in circumstances where AQMs are not deployed.

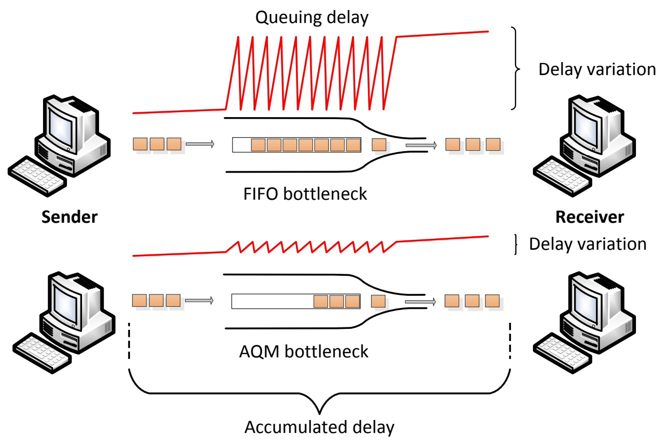


Fig. 14. The effect of using AQM on the delay signal

E. Quantifying Impact on Delay Signal

In this paper, we have highlighted a number of challenges when using the delay feedback signal for congestion control. However, our analysis is a qualitative work based on implementation details and outcomes observed by other researchers. To fully understand how these delay-based CC algorithms behave and compare under varying situations, a quantitative assessment of the impact on the delay signal is required.

This opens directions to further research to evaluate and compare the performance of delay-based CCs in different networks to better understand the behaviour of the delay feedback signal in different environments. The outcomes of such work should result in a clearer picture of the prevailing work with delay-based CC, and will help to develop better delay-based CC that satisfies the needs of specific applications and networks.

VII. Conclusions

During the last three decades, the Internet has become faster by many orders of magnitude and, at the same time, users have been deploying more internet based applications with a diverse mix of bandwidth and latency requirements. To meet the demands of the internet users and applications, academic and industry research have focused on improving the performance of TCP, the internet's dominant transport protocol.

Congestion control is a critical part of TCP, directly influencing transport performance. Consequently congestion control techniques have attracted a great deal of research attention. In this paper, we have surveyed a range of key congestion control algorithms that utilise the delay signal to infer the existence of congestion and/or use the measured delay as part of their congestion response behaviour.

Standard TCP CC is effective in protecting the network from collapse. However, this is not the sole concern of modern CC. Efficient capacity utilisation, low queueing delay and tolerance of non-congestion loss have become key requirements imposed by many applications and services. One of the biggest issues of the standard TCP is that it is unable to control the the latency caused by bottleneck queue congestion (or bufferbloat) due to use packet loss as a congestion feedback. This problem has serious impact on delay-sensitive applications such as video conference and multiplayer online gaming.

One solution to the bufferbloat problem is to manage queues using delay-focused AQM instead of DropTail management of FIFO queues. Many AQM proposed during the last two decades were not widely deployed because they were hard to optimally configure. Recently, new AQM strategies have re-emerged that are deployable, provide acceptable performance, while effectively controlling and limiting bottleneck queuing delay. Using ECN marking instead of packet dropping can provide an efficient queuing delay control without wasting network resources.

Delay-based congestion control interprets the delay in packet delivery (RTT or one-way delay) to early infer network congestion and control the congestion in an efficient manner without causing the buffer to bloat. In general, the delay-based approaches compare the delay signal with a threshold (constant or adaptive) or monitor the delay trend or gradient to infer congestion. Such techniques are capable of reducing bottleneck queue delay and packet loss rate and improving overall network performance.

However, using the delay signal as congestion feedback creates many challenges. Low fair share with standard TCP is one of the main issues of using the delay-based congestion control on the Internet. Some techniques primarily use the loss signal in their operations but utilise the delay signal as a secondary signal to improve the throughput, scalability and/or compatibility of the protocol. Others are designed to switch from delay mode to loss mode based on the type of the competing flows to provide butter compatibility with standard TCP.

Additionally, low-priority delay-based algorithms find that the unfairness is a desirable side effect which makes them work in the background without impacting other flows. Moreover, some CC protocols utilise delay in very limited parts of their operation such as calculating effective *cwnd* after packet loss event to enhance the performance in lossy environments.

Using delay-based congestion control in the Internet can raise link utilisation and reduce the effects of bufferbloat. There are many opportunities for researchers to conduct further studies to improve delay-based congestion control for specific or homogeneous environments. Finally, the coexistence of the delay-based approaches with emerging AQM techniques raises questions about what impact an AQM-based bottleneck can cause to the delay signal whether these bottleneck utilises ECN marking or packet dropping.

REFERENCES

[1] J. Postel, "Transmission Control Protocol," Internet Engineering Task Force, RFC 793, September 1981. [Online]. Available: https://tools.ietf.org/html/rfc793.txt

[2] ——, "Internet Protocol," Internet Engineering Task Force, RFC 791, September 1981. [Online]. Available: https://tools.ietf.org/html/rfc791.txt

[3] K. Ramakrishnan, "The Addition of Explicit Congestion Notification (ECN) to IP," Internet Engineering Task Force, RFC 3168, September 2001. [Online]. Available: https://tools.ietf.org/html/rfc3168

[4] D. Ros and M. Welzl, "Assessing LEDBAT's Delay Impact," *IEEE Communications Letters*, vol. 17, no. 5, pp. 1044–1047, May 2013.

[5] G. Armitage and N. Khademi, "Using delay-gradient TCP for multimedia-friendly 'background' transport in home networks," in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, Oct 2013, pp. 509–515.

[6] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low extra delay background transport (LEDBAT)," Internet Engineering Task Force, RFC 6817, December 2012. [Online]. Available: https://tools.ietf.org/html/rfc6817

[7] A. Venkataramani, R. Kokku, and M. Dahlin, "Tcp nice: A mechanism for background transfers," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 329–343, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844159

[8] C. Huitema, D. Havey, M. Olson, O. Ertugay, and P. Balasubramanian, "New Transport Advancements in the Anniversary Update for Windows 10 and Windows Server 2016," Microsoft, Jul 2016. [Online]. Available: https://goo.gl/ZfI8cG

[9] "TCP LEDBAT Implementation," Apple Inc. [Online]. Available: http://opensource.apple.com//source/xnu/xnu-3248.60.10/bsd/netinet/tcp_ledbat.c

[10] D. A. Hayes and G. Armitage, "Revisiting TCP congestion control using delay gradients," in *NETWORKING 2011*. Springer, 2011, pp. 328–341.

[11] R. Wang, M. Valla, M. Y. Sanadidi, B. K. F. Ng, and M. Gerla, "Efficiency/friendliness tradeoffs in TCP Westwood," in *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*, 2002, pp. 304–311.

[12] G. Hasegawa and M. Murata, "Survey on fairness issues in TCP congestion control mechanisms," *IEICE Transactions on Communications*, vol. 84, no. 6, pp. 1461–1472, 2001.

[13] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for tcp," *IEEE Communications Surveys Tutorials*, vol. 12, no. 3, pp. 304–342, Third 2010.

[14] D. Ros and M. Welzl, "Less-than-best-effort service: A survey of end-to-end approaches," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 898–908, Second 2013.

[15] 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, pp. 655–676, 2007. [Online]. Available: http://dx.doi.org/10.1002/wcm.524

[16] C. Callegari, S. Giordano, M. Pagano, and T. Pepe, *A Survey of Congestion Control Mechanisms in Linux TCP*. Cham: Springer International Publishing, 2014, pp. 28–42. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-05209-0_3

[17] S. Floyd, "Congestion Control Principles," Internet Engineering Task Force, RFC 2914, September 2000. [Online]. Available: https://tools.ietf.org/html/rfc2914

[18] J. Postel, "Internet Control Message Protocol," Internet Engineering Task Force, RFC 792, September 1981. [Online]. Available: https://tools.ietf.org/html/rfc791.txt

[19] F. Gont, "Deprecation of ICMP Source Quench Messages," Internet Engineering Task Force, RFC 6633, May 2012. [Online]. Available: https://tools.ietf.org/html/rfc6633

[20] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," *IEEE/ACM Trans. Netw.*, vol. 7, no. 4, pp. 458–472, Aug. 1999. [Online]. Available: http://dx.doi.org/10.1109/90.793002

[21] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," Internet Engineering Task Force, RFC 5681, September 2009. [Online]. Available: https://tools.ietf.org/html/rfc5681

[22] L. Brakmo and L. Peterson, "Tcp vegas: end to end congestion avoidance on a global internet," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 8, pp. 1465–1480, Oct 1995.

[23] R. J. La, J. Walrand, and V. Anantharam, *Issues in TCP vegas*. Electronics Research Laboratory, College of Engineering, University of California, 1999.

[24] B. Briscoe, "Flow rate fairness: Dismantling a religion," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 63–74, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1232919.1232926

[25] S. Floyd, A. Gurtov, and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," Internet Engineering Task Force, RFC 6582, April 2004. [Online]. Available: https://tools.ietf.org/html/rfc6582

[26] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Option," Internet Engineering Task Force, RFC 2018, October 1996. [Online]. Available: https://tools.ietf.org/html/rfc2018.txt

[27] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, Mar 2000, pp. 1157–1165 vol.3.

[28] D. Wei, P. Cao, S. Low, and C. EAS, "Tcp pacing revisited," in *Proceedings of IEEE INFOCOM*. Citeseer, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.2658&rep=rep1&type=pdf

[29] J. Sing and B. Soh, "TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency Links," in *Fourth IEEE International Symposium on Network Computing and Applications*, July 2005, pp. 73–82.

[30] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, pp. 40:40–40:54, Nov. 2011. [Online]. Available: http://doi.acm.org/10.1145/2063166.2071893

[31] F. Baker and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management," Internet Engineering Task Force, RFC 7567, July 2015. [Online]. Available: https://tools.ietf.org/html/rfc7567

[32] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 4, pp. 397–413, Aug 1993.

[33] S. Ryu, C. Rump, and C. Qiao, "Advances in internet congestion control," *IEEE Communications Surveys Tutorials*, vol. 5, no. 1, pp. 28–39, Third 2003.

[34] G. Thiruchelvi and J. Raja, "A Survey On Active Queue Management Mechanisms," *International journal of computer science and network security IJCSNS.*, vol. 8, no. 12, December 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.509.460&rep=rep1&type=pdf

[35] R. Adams, "Active queue management: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1425–1476, Third 2013.

[36] D. K. M. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," RFC 8289, Jan. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8289.txt

[37] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, July 2013, pp. 148–155.

[38] R. Pan, P. Natarajan, F. Baker, and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem," Internet Engineering Task Force, RFC 8033, February 2017. [Online]. Available: https://tools.ietf.org/html/rfc8033

[39] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm," Internet Engineering Task Force, RFC 8290, January 2018. [Online]. Available: https://tools.ietf.org/html/rfc8290

[40] R. Al-Saadi and G. Armitage, "Dummynet AQM v0.2 – CoDel, FQ-CoDel, PIE and FQ-PIE for FreeBSD's ipfw/dummynet framework," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 160418A, 18 April 2016. [Online]. Available: http://caia.swin.edu.au/reports/160418A/CAIA-TR-160418A.pdf

[41] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM computer communication review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.

[42] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008. [Online]. Available: http://doi.acm.org/10.1145/1400097.1400105

[43] Y. Tian, K. Xu, and N. Ansari, "Tcp in wireless environments: problems and solutions," *IEEE Communications Magazine*, vol. 43, no. 3, pp. S27–S32, March 2005.

[44] J. Andren, M. Hilding, and D. Veitch, "Understanding end-to-end Internet traffic dynamics," in *Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration. IEEE*, vol. 2, 1998, pp. 1118–1122 vol.2.

[45] J. Martin, A. Nilsson, and I. Rhee, "Delay-based congestion avoidance for tcp," *IEEE/ACM Trans. Netw.*, vol. 11, no. 3, pp. 356–369, Jun. 2003. [Online]. Available: http://dx.doi.org/10.1109/TNET.2003.813038

[46] S. Biaz and N. H. Vaidya, "Is the Round-trip Time Correlated with the Number of Packets in Flight?" in *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '03. New York, NY, USA: ACM, 2003, pp. 273–278. [Online]. Available: http://doi.acm.org/10.1145/948205.948240

[47] G. McCullagh and D. Leith, "Delay-based congestion control: Sampling and correlation issues revisited," *Hamilton Institute, National University of Ireland Maynooth, Tech. Rep*, 2008.

[48] R. S. Prasad, M. Jain, and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance," in *Proc. PFLDNet*, vol. 4, 2004.

[49] D. A. Hayes and D. Ros, "Delay-based congestion control for low latency," in *ISOC Workshop on Reducing Internet Latency, Sep*, 2013. [Online]. Available: http://www.internetsociety.org/sites/default/files/pdf/accepted/17_delay_cc_pos-v2.pdf

[50] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," Internet Engineering Task Force, RFC 1323, September 1992. [Online]. Available: https://tools.ietf.org/html/rfc1323

[51] A. Kuzmanovic and E. W. Knightly, "Tcp-lp: Low-priority service via end-point congestion control," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, pp. 739–752, Aug. 2006. [Online]. Available: http://dx.doi.org/10.1109/TNET.2006.879702

[52] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The Quest for LEDBAT Fairness," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, Dec 2010, pp. 1–6.

[53] K. Srijith, L. Jacob, and A. Ananda, "TCP Vegas-A: Improving the Performance of TCP Vegas," *Computer Communications*, vol. 28, no. 4, pp. 429 – 440, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366404003214

[54] A. J. Abu and S. Gordon, "Impact of Delay Variability on LEDBAT Performance," in *2011 IEEE International Conference on Advanced Information Networking and Applications*, March 2011, pp. 708–715.

[55] R. Jain, "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 5, pp. 56–71, Oct. 1989. [Online]. Available: http://doi.acm.org/10.1145/74681.74686

[56] D. Sisalem and A. Wolisz, "LDA+ TCP-friendly adaptation: A measurement and comparison study," in *Proc. NOSSDAV*, 2000, pp. 362–368. [Online]. Available: http://nossdav.org/2000/papers/23.pdf

[57] ——, "MLDA: a TCP-friendly congestion control framework for heterogeneous multicast environments," in *2000 Eighth International Workshop on Quality of Service. IWQoS 2000 (Cat. No.00EX400)*, 2000, pp. 65–74.

[58] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force, RFC 3550, jul 2003. [Online]. Available: https://tools.ietf.org/html/rfc3550.txt

[59] N. Khademi, M. Welzl, G. Armitage, and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)," Internet Engineering Task Force, RFC 8511, December 2018. [Online]. Available: https://tools.ietf.org/html/rfc8511

[60] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192

[61] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware Datacenter TCP (D2TCP)," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 115–126. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342388

[62] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 2157–2165.

[63] B. Briscoe, K. Schepper, and M. B. Braun, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture," Internet Engineering Task Force, Internet Draft, March 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-codel-07

[64] K. De Schepper, B. Briscoe, O. Bondarenko, and I. Tsang, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)," Internet Engineering Task Force, Internet Draft, July 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-codel-07

[65] S. Ha and I. Rhee, "Taming the elephants: New TCP slow start," *Computer Networks*, vol. 55, no. 9, pp. 2092 – 2110, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128611000363

[66] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms," Internet Engineering Task Force, RFC 5166, March 2008. [Online]. Available: https://tools.ietf.org/html/rfc5166

[67] A. Giessler, J. Haenle, A. König, and E. Pade, "Free buffer allocation - an investigation by simulation," *Computer Networks*

(1976), vol. 2, no. 3, pp. 191 – 208, 1978. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0376507578900284

[68] R. Jain, D.-M. Chiu, and W. R. Hawe, *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*. Eastern Research Laboratory, Digital Equipment Corporation Hudson, MA, 1984, vol. 38.

[69] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, "Dynamic behavior of slowly-responsive congestion control algorithms," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 263–274, Aug. 2001. [Online]. Available: http://doi.acm.org/10.1145/964723.383080

[70] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in the 4.3-tahoe bsd tcp congestion control algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 22, no. 2, pp. 9–16, Apr. 1992. [Online]. Available: http://doi.acm.org/10.1145/141800.141801

[71] C. Jin, D. Wei, S. H. Low, J. Bunn, H. D. Choe, J. C. Doylle, H. Newman, S. Ravot, S. Singh, F. Paganini, G. Buhrmaster, L. Cottrell, O. Martin, and W. chun Feng, "Fast tcp: from theory to experiments," *IEEE Network*, vol. 19, no. 1, pp. 4–11, Jan 2005.

[72] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast tcp: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006. [Online]. Available: http://dx.doi.org/10.1109/TNET.2006.886335

[73] S. Bhandarkar, A. L. N. Reddy, Y. Zhang, and D. Loguinov, "Emulating aqm from end hosts," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 349–360, Aug. 2007. [Online]. Available: http://doi.acm.org/10.1145/1282427.1282420

[74] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 537–550, Aug. 2015. [Online]. Available: http://doi.acm.org/10.1145/2829988.2787510

[75] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion Control for Low Latencies and High Throughput," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, Oct 2017, pp. 215–218.

[76] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of tcp reno and vegas," in *IEEE INFOCOM*, vol. 3. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1999, pp. 1556–1563.

[77] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet," in *Network Protocols, 2000. Proceedings. 2000 International Conference on*, 2000, pp. 177–186.

[78] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas revisited," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, Mar 2000, pp. 1546–1555 vol.3.

[79] K. Srijith, L. Jacob, and A. Ananda, "TCP Vegas-A: solving the fairness and rerouting issues of TCP Vegas," in *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, April 2003, pp. 309–316.

[80] L. Tan, C. Yuan, and M. Zukerman, "Fast tcp: fairness and queuing issues," *IEEE Communications Letters*, vol. 9, no. 8, pp. 762–764, Aug 2005.

[81] C. Jin, S. H. Low, and X. Wei, "Method and apparatus for network congestion control," Jul. 5 2011, uS Patent 7,974,195.

[82] C. Jin, S. Low, D. Wei, B. Wydrowski, A. Tang, and H. Choe, "Method and apparatus for network congestion control using queue control and one-way delay measurements," Apr. 6 2010, uS Patent 7,693,052. [Online]. Available: https://www.google.com/patents/US7693052

[83] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On designing improved controllers for aqm routers supporting tcp flows," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 3, April 2001, pp. 1726–1734 vol.3.

[84] "The Network Simulator - ns-2," ns-2. [Online]. Available: https://www.isi.edu/nsnam/ns/

[85] K. Kotla and A. L. N. Reddy, "Making a delay-based protocol adaptive to heterogeneous environments," in *2008 16th Interntional Workshop on Quality of Service*, June 2008, pp. 100–109.

[86] L. Budzisz, R. Stanojevic, A. Schlote, R. Shorten, and F. Baker, "On the fair coexistence of loss- and delay-based TCP," in *17th International Workshop on Quality of Service, IWQoS 2009, Charleston, South Carolina, USA, 13-15 July 2009.*, 2009, pp. 1–9. [Online]. Available: https://doi.org/10.1109/IWQoS.2009.5201387

[87] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: The New BitTorrent Congestion Control Protocol," in *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, Aug 2010, pp. 1–6.

[88] "uTorrent Transport Protocol," Bittorrent.org. [Online]. Available: http://www.bittorrent.org/beps/bep_0029.html

[89] "libutp - uTorrent Transport Protocol implementation," BitTorrent Inc., 2010. [Online]. Available: https://github.com/bittorrent/libutp

[90] J. Schneider, J. Wagner, R. Winter, and H. J. Kolbe, "Out of my way - evaluating Low Extra Delay Background Transport in an ADSL access network," in *Teletraffic Congress (ITC), 2010 22nd International*, Sept 2010, pp. 1–8.

[91] R. Al-Saadi, G. Armitage, and J. But, "Characterising LEDBAT Performance Through Bottlenecks Using PIE, FQ-CoDel and FQ-PIE Active Queue Management," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, Oct 2017, pp. 278–285.

[92] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "Ecn or delay: Lessons learnt from analysis of dcqcn and timely," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 2016, pp. 313–327.

[93] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: yet another highspeed TCP," in *Proc. PFLDnet*, vol. 7, 2007, pp. 37–42. [Online]. Available: http://www.gdt.id.au/~gdt/presentations/2010-07-06-questnet-tcp/reference-materials/papers/baiocchi+castellani+vacirca-yeah-tcp-yet-another-highspeed-tcp.pdf

[94] . Budzisz, R. Stanojevic, A. Schlote, F. Baker, and R. Shorten, "On the fair coexistence of loss- and delay-based tcp," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1811–1824, Dec 2011.

[95] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*. USENIX Association, apr 2018. [Online]. Available: https://www.usenix.org/system/files/conference/nsdi18/nsdi18-arun.pdf

[96] P. Goyal, A. Narayan, F. Cangialosi, D. Raghavan, S. Narayana, M. Alizadeh, and H. Balakrishnan, "Elasticity detection: A building block for delay-sensitive congestion control," *arXiv preprint arXiv:1802.08730*, 2018.

[97] S. Floyd, "HighSpeed TCP for large congestion windows," Internet Engineering Task Force, RFC 3649, December 2003. [Online]. Available: https://tools.ietf.org/html/rfc3649

[98] T. Kelly, "Scalable tcp: Improving performance in highspeed wide area networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, Apr. 2003. [Online]. Available: http://doi.acm.org/10.1145/956981.956989

[99] A. H. David and G. Armitage, "Improved coexistence and loss tolerance for delay based TCP congestion control," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, Oct 2010, pp. 24–31.

[100] T. C. Tangenes, D. A. Hayes, A. Petlund, and D. Ros, "Evaluating CAIA Delay Gradient as a Candidate for Deadline-Aware Less-than-Best-Effort Transport," *Workshop on Future of Internet Transport (FIT 2017), Stockholm*, june 2017. [Online]. Available: http://dl.ifip.org./db/conf/networking/networking2017/1570350870.pdf

[101] N. Hu and P. Steenkiste, "Estimating available bandwidth using packet pair probing," CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 2002.

[102] K. Lai and M. Baker, "Nettimer: A tool for measuring bottleneck link bandwidth." in *USITS*, vol. 1, 2001, pp. 11–11.

[103] V. Jacobson, "Pathchar: A tool to infer characteristics of internet paths," 1997.

[104] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," in *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4. ACM, 2000, pp. 283–294.

[105] R. King, R. Baraniuk, and R. Riedi, "TCP-Africa: an adaptive and fair rapid increase rule for scalable TCP," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3, March 2005, pp. 1838–1848 vol. 3.

[106] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–12.

[107] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Y. Sanadidi, and M. Roccetti, *TCP Libra: Exploring RTT-Fairness for TCP*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1005–1013. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72606-7_86

[108] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks," *Performance Evaluation*, vol. 65, no. 6, pp. 417–440, 2008.

[109] "Description of a hotfix that adds Compound TCP (CTCP) support to computers that are running Windows Server 2003 or a 64-bit version of Windows XP," Microsoft Corporation. [Online]. Available: https://support.microsoft.com/en-gb/help/949316/description-of-a-hotfix-that-adds-compound-tcp-ctcp-support-to-computeCorporation

[110] P. Balasubramanian, "Updates on Windows TCP," Microsoft, nov 2017. [Online]. Available: https://datatracker.ietf.org/meeting/100/materials/slides-100-tcpm-updates-on-windows-tcp

[111] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi, "CapProbe: A Simple and Accurate Capacity Estimation Technique," in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '04. New York, NY, USA: ACM, 2004, pp. 67–78. [Online]. Available: http://doi.acm.org/10.1145/1015467.1015476

[112] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '01. New York, NY, USA: ACM, 2001, pp. 287–297. [Online]. Available: http://doi.acm.org/10.1145/381677.381704

[113] L. A. Grieco and S. Mascolo, *TCP Westwood and Easy RED to Improve Fairness in High-Speed Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 130–146. [Online]. Available: http://dx.doi.org/10.1007/3-540-47828-0_9

[114] R. Wang, M. Valla, M. Y. Sanadidi, and M. Gerla, "Adaptive bandwidth share estimation in TCP Westwood," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 3, Nov 2002, pp. 2604–2608 vol.3.

[115] G. Yang, R. Wang, M. Y. Sanadidi, and M. Gerla, "TCPW with bulk repeat in next generation wireless networks," in *Communications, 2003. ICC '03. IEEE International Conference on*, vol. 1, May 2003, pp. 674–678 vol.1.

[116] S. M. ElRakabawy and C. Lindemann, "A Practical Adaptive Pacing Scheme for TCP in Multihop Wireless Networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 4, pp. 975–988, Aug 2011.

[117] V. Konda and J. Kaur, "RAPID: Shrinking the Congestion-Control Timescale," in *IEEE INFOCOM 2009*, April 2009, pp. 1–9.

[118] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Commun. ACM*, vol. 60, no. 2, pp. 58–66, Jan. 2017. [Online]. Available: http://doi.acm.org/10.1145/3009824

[119] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 25–38, Apr. 2004. [Online]. Available: http://doi.acm.org/10.1145/997150.997155

[120] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 21, no. 4, pp. 133–147, Aug. 1991. [Online]. Available: http://doi.acm.org/10.1145/115994.116006

[121] H. Shimonishi, M. Y. Sanadidi, and M. Gerla, "Improving efficiency-friendliness tradeoffs of TCP in wired-wireless combined networks," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 5, May 2005, pp. 3548–3552 Vol. 5.

[122] R. Wang, K. Yamada, M. Y. Sanadidi, and M. Gerla, "Tcp with sender-side intelligence to handle dynamic, large, leaky pipes," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 235–248, Feb 2005.

[123] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, Oct 2017, pp. 1–10.

[124] J. Karlsson, A. Kassler, and A. Brunstrom, "Impact of packet aggregation on TCP performance in Wireless Mesh Networks," in *2009 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks Workshops*, June 2009, pp. 1–7.

[125] F. Hui and P. Mohapatra, "Experimental characterization of multi-hop communications in vehicular ad hoc network," in *Proceedings of the 2Nd ACM International Workshop on Vehicular Ad Hoc Networks*, ser. VANET '05. New York, NY, USA: ACM, 2005, pp. 85–86. [Online]. Available: http://doi.acm.org/10.1145/1080754.1080770

[126] A. P. Silva, S. Burleigh, C. M. Hirata, and K. Obraczka, "A survey on congestion control for delay and disruption tolerant networks," *Ad Hoc Networks*, vol. 25, pp. 480 – 494, 2015, new Research Challenges in Mobile, Opportunistic and Delay-Tolerant Networks Energy-Aware Data Centers: Architecture, Infrastructure, and Communication. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870514001668

[127] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Taht, "Fighting the bufferbloat: On the coexistence of AQM and low priority congestion control," *Computer Networks*, vol. 65, pp. 255 – 267, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614000188

Rasool Al-Saadi earned a B.Sc. and M.Sc. in computer science from University of Baghdad, Iraq, in 2002 and 2005 respectively. He is currently pursuing the Ph.D. degree with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia. He has worked a lecturer at Al-Nahrain University, Baghdad, Iraq since 2005. His research interests include data transport protocols, low-priority and hybrid congestion control, and active queue management.

Grenville Armitage is Engineering Manager of the Open Connect Appliance Transport Protocol R&D group at Netflix Inc, and Adjunct Professor at Swinburne University of Technology, Australia. He earned a B.Eng. in electrical engineering (Hons) in 1988 and a Ph.D. in electronic engineering in 1994, both from the University of Melbourne, Australia. Between 1994 and 2001 he worked for Bellcore and Lucent Bell Labs in technical R&D and product marketing roles, based in New Jersey and California. Between 2002 and mid-2018 he was employed by Swinburne University of Technology (SUT), Australia, as an associate professor then full professor of telecommunications engineering, was founding director of SUT's Centre for Advanced Internet Architectures (2002-2017), and founding head of SUT's Internet For Things (I4T) Research Group (2017). He authored "Quality of Service In IP Networks: Foundations for a Multi-Service Internet" (Macmillan, April 2000) and co-authored "Networking and Online Games 'Understanding and Engineering Multiplayer Internet Games'" (Wiley, April 2006). He is a member of IEEE, ACM and ACM SIGCOMM. Grenville has been active in the data networking research community for over two decades, pursuing evidence-based applied research and development.

Jason But earned a B.Eng. (Elec)(Hons) and a BSc (CompSci) in 1995 from the University of Melbourne, Australia and a PhD in Telecommunications Engineering in 2004 from Monash University, Australia. Since 2004 he has held positions as a Research Fellow and later as a Lecturer and Senior Lecturer at Swinburne University of Technology, conducting research within the Centre for Advanced Internet Architectures (2004-2017) and the Internet for Things Research Group (2017-). His research interests are in the perceived performance of networked applications, QoS and performance evaluation, Network Protocols, and Software Defined Networking.

Philip Branch is an Associate Professor in Computer Systems Engineering at Swinburne University of Technology. He has research and teaching interests in Network Security, Wireless Networks, Internet of Things, and applications of Machine Learning. He has a PhD from Monash University, a BSc and MTech from the University of Tasmania, a GradCert in Teaching from Swinburne and a number of industry certifications. He has over eighty refereed publications.