Reducing Internet Latency: A Survey of Techniques and Their Merits

Bob Briscoe, Anna Brunstrom, Andreas Petlund, David Hayes, David Ros, Ing-Jyh Tsang, Stein Gjessing, Gorry Fairhurst, Carsten Griwodz, and Michael Welzl

***Abstract*—Latency is increasingly becoming a performance bottleneck for Internet Protocol (IP) networks, but historically, networks have been designed with aims of maximizing throughput and utilization. This paper offers a broad survey of techniques aimed at tackling latency in the literature up to August 2014, as well as their merits. A goal of this work is to be able to quantify and compare the merits of the different Internet latency reducing techniques, contrasting their *gains* in delay reduction versus the *pain* required to implement and deploy them. We found that classifying techniques according to the *sources of delay* they alleviate provided the best insight into the following issues: 1) The structural arrangement of a network, such as placement of servers and suboptimal routes, can contribute significantly to latency; 2) each interaction between communicating endpoints adds a Round Trip Time (RTT) to latency, particularly significant for short flows; 3) in addition to base propagation delay, several sources of delay accumulate along transmission paths, today intermittently dominated by queuing delays; 4) it takes time to sense and use available capacity, with overuse inflicting latency on other flows sharing the capacity; and 5) within end systems, delay sources include operating system buffering, head-of-line blocking, and hardware interaction. No single source of delay dominates in all cases, and many of these sources are spasmodic and highly variable. Solutions addressing these sources often both reduce the overall latency and make it more predictable.**

***Index Terms*—Data communication, networks, Internet, performance, protocols, algorithms, standards, cross-layer, comparative evaluation, taxonomy, congestion control, latency, queuing delay, bufferbloat.**

I. Introduction

Many, if not most, Internet Protocol (IP) networks and protocols have traditionally been designed with optimization of throughput or link utilization in mind. Such a focus on "bandwidth" may well be justified for bulk-data transfer, or more generally for applications that do not require timeliness in their data delivery. However, nowadays the quality of experience delivered by many applications depends on the delay to complete short data transfers or to conduct real-time conversations, for which adding bandwidth makes little or no difference. As a result, latency in the current Internet has been gaining visibility as a truly critical issue that impairs present-day applications, and that may hinder the deployment of new ones. It is therefore important to: (a) understand the root causes of latency, and (b) assess the availability of solutions, deployed or not, and their expected contribution to lowering end-to-end latency. This paper seeks to address these questions. We offer a broad survey of techniques aimed at tackling Internet latency up to August 2014, classifying the techniques according to the *sources* of delay that they address, i.e., where delays arise along the communications chain. To decide on the best classification system, we tried a number of alternative systems: classifying by sources of delay was the easiest to understand and led to the fewest gaps and least overlap. We also attempt to quantify the merits of a selection of the most promising techniques. We decided to focus on reduction in delay and ease of deployment, which loosely represent the main tradeoff between benefit and cost ('gain vs. pain'). The benefits of any technique are highly scenario-dependent, so we carefully chose a set of scenario parameters that would be amenable to visual comparison across an otherwise complex space.

A. Importance of Latency to Applications

Latency is a measure of the responsiveness of an application; how instantaneous and interactive it feels, rather than sluggish and jerky. In contrast to bandwidth, which is the rate at which bits can be delivered, latency is the time it takes for a single critical bit to reach the destination, measured from when it was first required. This definition may be stretched for different purposes depending on which bit is 'critical' for different applications, the main categories being:

1) Real-time interaction, where every 'chunk' of data produced by an end-point is unique and of equal importance and needs to be delivered as soon as possible, for example an on-line game, or an interactive video conference (the 'critical bit' is therefore the last bit of each 'chunk').
2) Start-up latency, where the time to begin a service is most important, as at the start of a video stream (the 'critical bit' is the first bit of data).
3) Message completion time, where the time to complete a whole (often small) transmission is most important, for example downloading Javascript code to start an application (the 'critical bit' is the last bit of the message).

Manuscript received May 9, 2014; revised September 1, 2014; accepted October 17, 2014. Date of publication November 26, 2014; date of current version August 19, 2016. This work was supported in part by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) Project (ICT-317700). The views expressed are solely those of the authors.

B. Briscoe is with BT, Ipswich IP5 3RE, U.K.
A. Brunstrom is with Karlstad University, 651 88 Karlstad, Sweden.
A. Petlund, D. Ros, and C. Griwodz are with Simula Research Laboratory AS, 1364 Fornebu, Norway.
D. Hayes, S. Gjessing, and M. Welzl are with the University of Oslo, 0316 Oslo, Norway.
I.-J. Tsang is with Bell Labs, Alcatel-Lucent, 2018 Antwerpen, Belgium.
G. Fairhurst is with the University of Aberdeen, AB24 3FX Aberdeen, U.K.

Digital Object Identifier 10.1109/COMST.2014.2375213

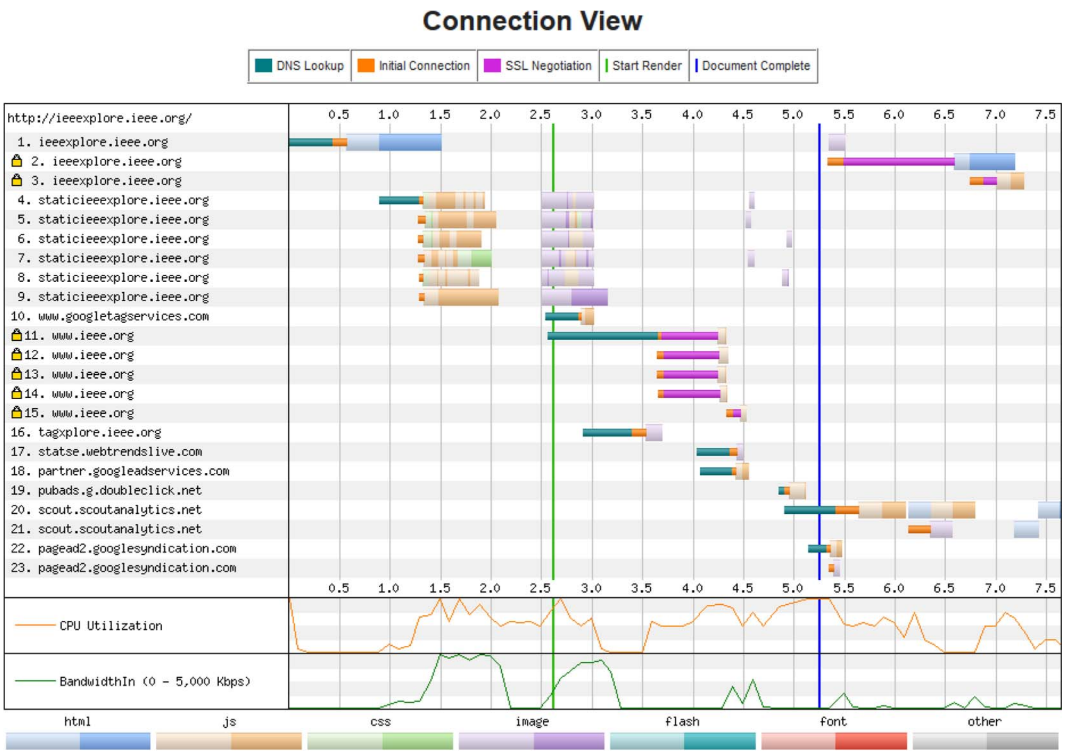


Fig. 1. Waterfall diagram showing the timing of download of an apparently uncluttered example Web page (ieeexplore.ieee.org), actually comprising over one hundred objects, transferred over 23 connections needing 10 different DNS look-ups. The horizontal scale is in seconds. This access was from Stockholm, Sweden, over a 28 ms RTT 5 Mb/s down 1 Mb/s up cable access link, using Internet Explorer v8 without any prior cache warming. Source: www.webpagetest.org.

An important characteristic of latency is that it is additive in nature and accumulates over the communication session or application task. Distributed systems involving machine to machine interactions, e.g., Web services, consist of long sequences of automated interactions in between each human intervention. Therefore even slight unnecessary delay per transfer adds up to considerable overall delay. For instance, Fig. 1 shows the connections involved in downloading an apparently uncluttered Web page (ieeexplore.ieee.org). This unremarkable example is fairly typical of many Web pages. Closer examination shows that the critical path of serial dependencies consists of about half a dozen short TCP connections, each starting with a 3-way handshake and each preceded by a DNS look-up, adding up to at least six back-and-forth messages each, and totaling nearly forty transfers in series. It might be considered that delays up to say 50 ms are so close to the bounds of human perception that they are not worth removing. However, adding 40×50 ms would delay completion of this example page by about 2 seconds. Experiencing such unnecessary delay on every click during a browsing session makes the experience unnecessarily intermittent and unnatural.

In 2009 a team from Microsoft found that artificially introducing 500 ms extra delay in the response from the Bing search engine translated to 1.2% less advertising revenue. Google experimented with injecting 400 ms delay before they returned their search page to a subset of their users. Initial searches declined linearly over time, dropping by 0.76% after 6 weeks and continuing linearly after that. Interestingly, once the artificially introduced delay was removed, it took a similar period to linearly regain the original demand. These results were presented in a joint Microsoft-Google presentation on the value of reducing Web delay [1]. Google's VP for search pointed out that if Google's experiment had been conducted on their whole customer base, losing 0.75% of their 2009 revenue would have lost them $75M for the year [2].

Certain applications suffer more when there is a large variation in latency (jitter). This includes time-stepped applications such as voice or applications with real-time interaction, such as on-line games [3]. In general, latency is characterized by its distribution function, and the relative importance of the higher order moments is application dependent.

B. Scope

We restrict our scope to generic techniques that are each applicable to a wide range of ways the Internet could be used in its role as a public network. Still, a few promising techniques currently applicable only in private networks are included when we see that they may be adapted to, or inspire solutions for, the public Internet. We also draw a fairly arbitrary line to rule out more specialist techniques. For instance, we include the delay that certain applications (e.g., VoIP) experience when initializing a session through a network address translator, but we exclude initialization delay for other more specialist middleboxes and applications.

We restrict our survey to sources of latency when the Internet is working as it should. This excludes significant causes of latency such as natural disasters, accidental misconfiguration, tolerance of network entities to faults and failures, and malicious denial of service attacks—such causes can induce excessive latency in the various sources we discuss, but they require their own specialized treatment.

C. Paper Outline

The remainder of the paper is organized as follows: Sections II–VI contain the main part of the paper: the survey of available techniques for reducing communication latency. The organization of these sections is outlined next. Then Section VII illustrates how several techniques for reducing latency can be combined into integrated solutions. For instance, this section highlights the use of WAN accelerators and new protocols, such as SPDY and QUIC. In Section VIII we justify having chosen sources of delay as the organizing principle of our survey. We briefly introduce alternative classification systems that we considered, and outline their pros and cons. In Section IX we seek to quantify the gains offered by some key techniques from the survey, relating this to a set of representative communication scenarios. We provide a visualization of the tradeoff between this gain and the likely difficulty to deploy each technique. Finally, Section X draws conclusions.

D. Organization of Survey

There are clearly many possible ways in which we could have organized this survey. We have chosen an organization that presents techniques based on an analysis of the different sources of delay encountered during a communication session. As with any classification scheme, not all techniques map perfectly onto the resulting structure, but we have found this taxonomy to be the most useful organization for highlighting the various causes of latency in the Internet and furthering an understanding of how latency can be reduced.

Fig. 2 illustrates the organization of the main survey part of this paper. The higher levels in the tree in Fig. 2 represent sources of delay and the lowest level corresponds to families of techniques for reducing this delay. The sources of delay are classified into five main categories: structural delays, interaction between endpoints, delays along transmission paths, delays related to link capacities, and intra-end-host delays.

Structural delays (Section II) arise from the structure of the network or the communication path that is used. This, for instance, includes delays due to a suboptimal placement of servers or content, and delays due to using suboptimal routes. Structural delays and techniques to reduce them are illustrated in yellow in Fig. 2.

Delays resulting from the interaction between endpoints (Section III) include delays due to transport initialization and secure session initialization, as well as delays from recovering lost packets and from message-aggregation techniques. Delays from the interaction between endpoints and techniques to reduce them are illustrated in light orange in Fig. 2.

Delays along transmission paths (Section IV) captures the delays that may be encountered as data travels between a sender and a receiver. This, for instance, includes propagation delay and delay due to queuing in network nodes. Delays along transmission paths and techniques to reduce them are illustrated in green in Fig. 2.

Delays related to link capacities (Section V) include both delays resulting from sharing limited capacity and delays from protocol inefficiencies that under-utilize capacity and therefore communication takes longer than necessary. Delays related to link capacities and techniques to reduce them are illustrated in blue in Fig. 2.

Intra-end-host delays (Section VI) are delays that occur internally within host endpoints. This includes delays due to buffering in the transport protocol stack and delays within the operating system. Intra-end-host delays are illustrated in red in Fig. 2.

While we have organized the presentation based on sources of delay, it should be noted that similar principles can be shared by solutions that address different sources of delay and are applied at different layers. For instance, principles and solutions used to reduce queuing delay along a transmission path can also be used to reduce the delay due to buffering within an endpoint.

II. Structural Delays

Internet communication relies on interactions between a set of endpoint systems. The placement of the software components, such as servers, caches, databases and proxies in relation to a client endpoint can have a significant impact on the application latency experienced by the client. The type of application also imposes restrictions on the set of methods a systems architect can use to minimize delay. Finally, when a systems architect has chosen a placement scheme for the components, the strategies used for accessing data need to be wisely chosen to minimize delays. Given client-server is a common arrangement, this section mostly focuses on ways to minimize latency by optimizing the placement of services and the use of data access techniques between Internet hosts that reduce the delay experienced by a user (client) accessing a server or a server backend system.

A. Sub-Optimal Routes/Paths

As illustrated in Fig. 3, the path used by the user endpoint to receive a particular service may utilize a range of routers, layer 2 communications services, servers, caches, and proxies. When a packet travels between two of these entities, the total latency is the sum of the propagation latencies of the links that form the path, plus the latency inside all the network devices (switches, routers, or network middleboxes) along the path. This path latency is discussed in Section IV.

The selected path is usually determined by a routing protocol—implemented by methods such as Multiprotocol Label Switching (MPLS [4]), Border Gateway Protocol (BGP [5]) or Open Shortest Path First (OSPF [6]). These routing protocols are typically configured with policies to optimize the choice of path, dynamically exchange information to determine the best path, or use a combination of the two to optimize a metric (such as number of hops, or lowest "cost," often expressed as the inverse of the maximum link capacity).

Routing methods typically offer robustness to link failure, but protocols that take congestion into account are confined to the research domain [7], probably due to unresolved concerns over stability. Hence, a minimal cost route based on link capacities or link lengths might not be the path with the current shortest latency, especially if this path is shared with other traffic that adds queuing delay. Although methods such as Equal Cost Multipath Routing (ECMP [8]) allow simultaneous use of more

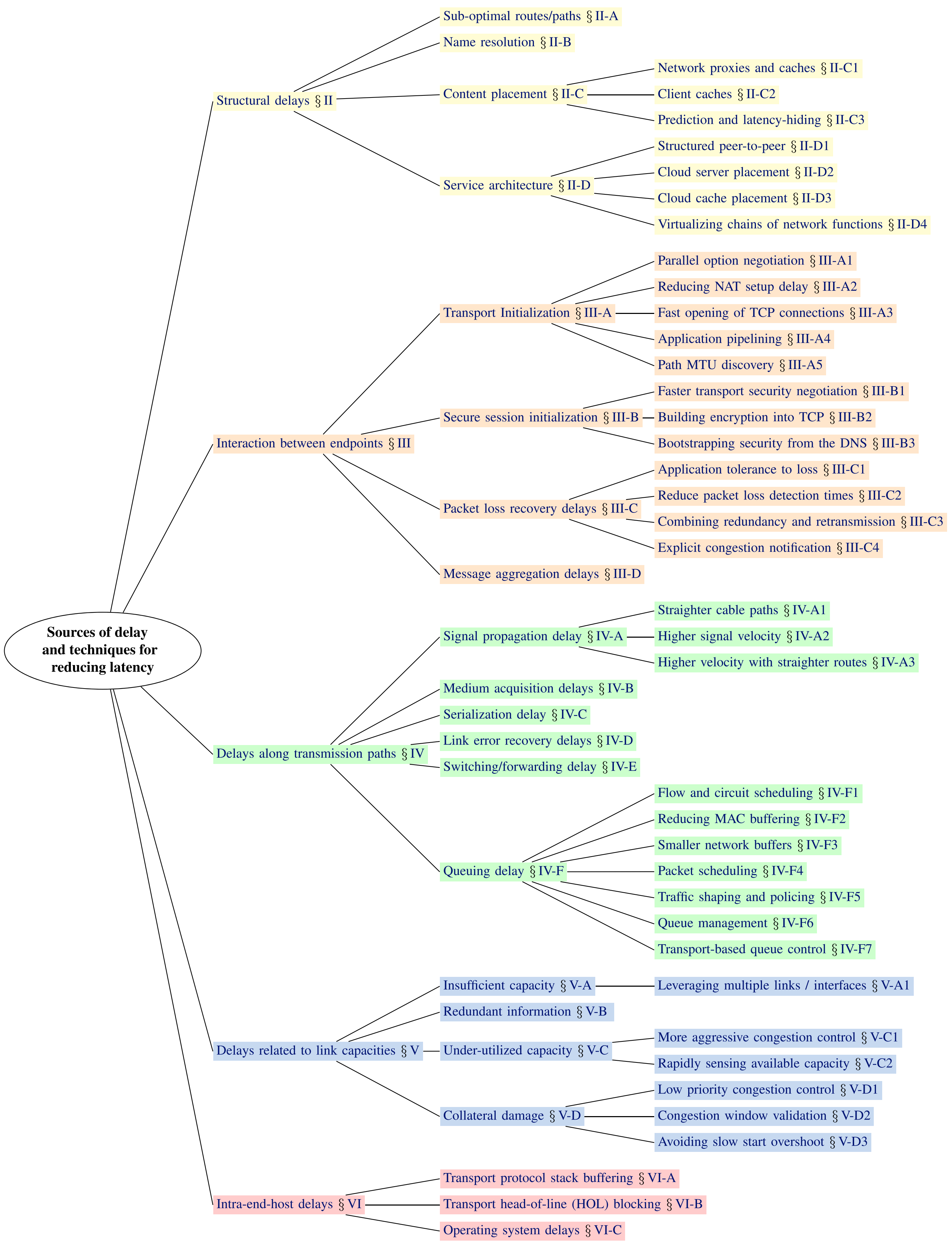

Sources of delay and techniques for reducing latency
Structural delays § II
Sub-optimal routes/paths § II-A
Name resolution § II-B
Content placement § II-C
Network proxies and caches § II-C1
Client caches § II-C2
Prediction and latency-hiding § II-C3
Service architecture § II-D
Structured peer-to-peer § II-D1
Cloud server placement § II-D2
Cloud cache placement § II-D3
Virtualizing chains of network functions § II-D4
Interaction between endpoints § III
Transport Initialization § III-A
Parallel option negotiation § III-A1
Reducing NAT setup delay § III-A2
Fast opening of TCP connections § III-A3
Application pipelining § III-A4
Path MTU discovery § III-A5
Secure session initialization § III-B
Faster transport security negotiation § III-B1
Building encryption into TCP § III-B2
Bootstrapping security from the DNS § III-B3
Packet loss recovery delays § III-C
Application tolerance to loss § III-C1
Reduce packet loss detection times § III-C2
Combining redundancy and retransmission § III-C3
Explicit congestion notification § III-C4
Message aggregation delays § III-D
Delays along transmission paths § IV
Signal propagation delay § IV-A
Straighter cable paths § IV-A1
Higher signal velocity § IV-A2
Higher velocity with straighter routes § IV-A3
Medium acquisition delays § IV-B
Serialization delay § IV-C
Link error recovery delays § IV-D
Switching/forwarding delay § IV-E
Queuing delay § IV-F
Flow and circuit scheduling § IV-F1
Reducing MAC buffering § IV-F2
Smaller network buffers § IV-F3
Packet scheduling § IV-F4
Traffic shaping and policing § IV-F5
Queue management § IV-F6
Transport-based queue control § IV-F7
Delays related to link capacities § V
Insufficient capacity § V-A
Leveraging multiple links / interfaces § V-A1
Redundant information § V-B
Under-utilized capacity § V-C
More aggressive congestion control § V-C1
Rapidly sensing available capacity § V-C2
Collateral damage § V-D
Low priority congestion control § V-D1
Congestion window validation § V-D2
Avoiding slow start overshoot § V-D3
Intra-end-host delays § VI
Transport protocol stack buffering § VI-A
Transport head-of-line (HOL) blocking § VI-B
Operating system delays § VI-C

Fig. 2. Techniques for reducing latency organized by sources of delay.

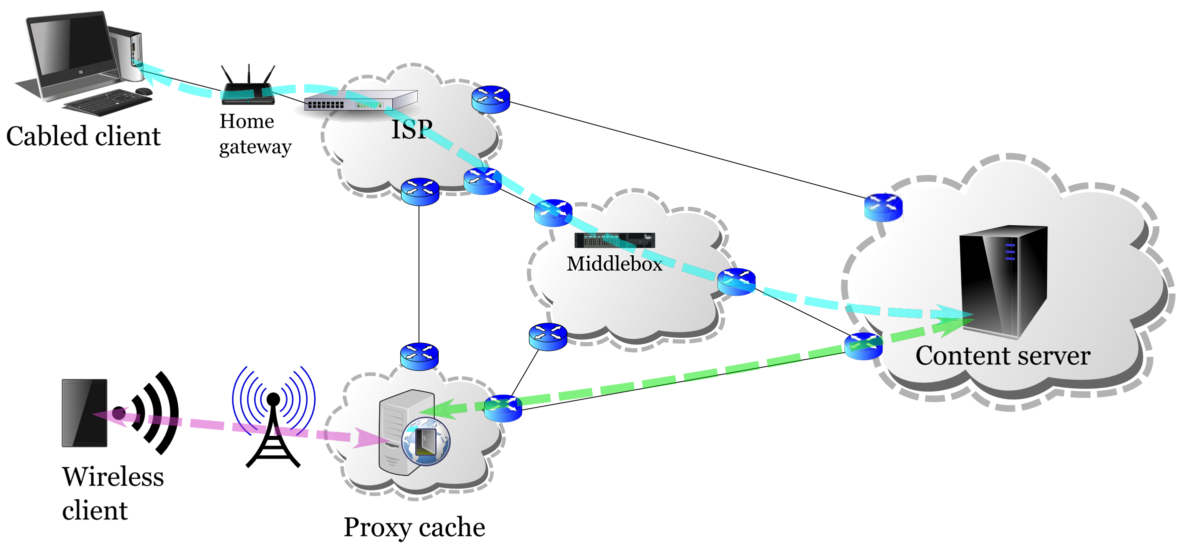


Fig. 3. Example of clients connecting to a server over the Internet. When a client requests a service over the Internet the connection may have to traverse a wide range of different systems. Different access network technologies have different requirements and limitations, and network routing decisions based on peering agreements may lead to sub-optimal routes from a latency perspective. Although transparent to the user, the connection might actually be intercepted by a middlebox or served by a proxy cache. At any intermediate node, there may be buffering or processing delays that impacts the latency experienced by the user.

than one path—enabling the latency to be reduced by sharing the capacity of multiple parallel links—a single flow is typically assigned to only one path, which can result in two different flows between the same pair of endpoints encountering very different latency.

In Multi Topology Routing (MTR), each router has multiple forwarding tables, one for each class of service [9]. If link latency is one cost metric, then one class of service may forward along the path with the lowest delay. A solution may be offered by a multi-path routing protocol that can find several paths between source and destination [10], dynamically routing latency-sensitive traffic over the lowest delay path.

Paths may also be controlled at the transport layer. In Multipath TCP (MPTCP [11], [12]), several paths are created between the sender and the receiver, and a sender can then choose which path (or subflow) to use for sending. Unlike network-based techniques, the stability bounds of MPTCP have been derived [13]. The RTT of each subflow is known by the congestion control protocol on the sender side, which could be used by MPTCP to select the lowest latency path. Apple's natural speech support service, Siri, uses MPTCP mainly for resilience, but it is also hard-coded to prefer WiFi over cellular given the likelihood of lower latency [14]. A future MPTCP API is envisioned to give the application the ability to choose a path with low latency and low jitter [15]. MPTCP is further described in Section V-A.

Since routing decisions between networks are often constrained by the economic guidelines of peering agreements between Internet Service Providers (ISP), the final chosen path may not have the shortest latency, even when using a routing latency metric. When a system architect wants to ensure that a certain path is used so that latency for the service is reduced, they can buy a dedicated tunneling service from their ISP, reducing the chance of incurring extra latency from routing. Another option is to use overlay hosts [16] to dynamically identify alternative routes to endpoints.

B. Name Resolution

Most applications do not directly use the Internet addresses that are used for routing, but instead rely on a name resolution step, at least once before a client can request the actual service for the first time. In name resolution, a client process called the resolver retrieves addresses for a given domain name by requesting a record from the Domain Name Service (DNS). An earlier DNS response is usually cached by a resolver for a period of time. If the record is not cached, a request adds a latency of at least the RTT between the requesting end-systems and the local DNS server. This local DNS server also usually caches responses from the DNS system. When a record cannot be resolved locally, a referral is forwarded upwards along the DNS hierarchy, and potentially down again along the chain of authoritative DNS servers for the specific name.

Jung *et al.* [17] surveyed the literature and studied DNS resolution from the US and South Korea. They found that roughly 85% of all requests were served from local DNS servers. For 10% of all incoming TCP connections, servers would additionally perform a reverse DNS lookup before allowing a connection. Approximately 10% (US) and 25% (South Korea) of all name lookups took more than 1 s, with more than 5% of all lookups from South Korea taking more than 10 s. There can therefore be a significant variation in lookup time. The time for a local DNS server to resolve a request required more than 1 s in 8% of cases, while more than 45% of DNS lookups took more than 1 s if 2 referrals were required.

Jung *et al.* found that names were Zipf-distributed across all requests. This implies that an arbitrary enlargement of the caches would not significantly increase the hit rate of the local DNS server, because a DNS record expires after timeout. Although timeout values are lowest for highly popular sites (5 min is typical for websites that use DNS-based load-leveling strategies), most cache misses are still due to lookup operations for infrequently requested names. Ager *et al.* [18] presented further analysis of load-leveled local DNS servers. Some ISPs use random load-leveling, which then loses the benefits of caching.

DNS pre-fetching is a method that can reduce latency by attempting to resolve domain names before they are requested [19], [20]. For example, a web browser may attempt to resolve names when a web page is received and before a user chooses to follow a specific URL. This is based on analyzing the content or using explicit "pre-fetch" meta tags. This method can also be used to fill the cache of a DNS proxy when a web proxy is used and the web page is not encrypted. The cost of DNS pre-fetching is that it generates additional network traffic, thus consuming resources and potentially contributing to latency due to contention.

Reducing DNS lookup latency is only one possible gain. Often the DNS is used to direct the following data connections of a client to a preferred server, which may be geographically or topologically closer, have lower estimated latency to the client, or be less heavily loaded than others. An example of a system that provides such context-specific DNS responses is Akamai's Global Traffic Manager [21]. This can significantly reduce latency. Otto *et al.* [22] found that the benefit depends on the distance between the resolver and the local DNS server. They found that 27% of ISPs clients' resolvers were topologically distant "far-away" DNS servers, increasing the median time for starting a TCP connection, for example by 50% accessing Akamai content. This optimization relies on locating the client's resolver. However, a current trend is to anonymize the client location by bypassing the local ISP's DNS server and using open DNS resolvers with arbitrary physical locations. Ager *et al.* [18] found that such resolvers fail to resolve to servers in the local network, preventing optimization. Otto *et al.* [22] clarify that the effect of using open DNS resolvers is identical to using a far-away local DNS server assigned by an ISP, i.e., a 50% median increase in the DNS latency.

C. Content Placement

The proximity of the content to the consumer is one of the most important factors determining the experienced latency. Services confined to run at a single location may experience high loads. This can increase the server response time, and/or result in increased queuing delay on the path to the server, leading to high finishing times (Section IV). Also, a well-built hierarchy of content may be necessary to make a popular service scale to the demand. Considering which strategies to use for placing the content to be served is therefore important to anyone providing applications over the Internet.

1) Network Proxies and Caches: The caching techniques used by the DNS (Section II-B) may also be used for application data, allowing replicas of content to be served from other network devices to reduce the completion time of transfers.

The simplest method is *passive caching*. This distributes a replica of the content to a node closer to the client. All caches have a limited storage and typically implement a content-agnostic object replacement scheme. Although this will reduce the access time for data available at the cache, it will sometimes be necessary to contact the data source when there is a cache miss, or content is uncachable. A cache miss increases the response time, although the additional time is often small compared to the RTT between the server and client. Caches may be organized in tiers or layers and cache subtypes can be identified (e.g., in the context of Web caching [23]).

A *proxy cache* is a network device that is typically located close to a population of clients. It can be either a transparent or a non-transparent (configured) proxy, and requires application-level processing of each request. Proxy caches can reduce all sources of latency, but they offload the content providers in an unpredictable manner. Podlipnig and Böszörmenyi surveyed object replacement strategies for web objects [25] and for dedicated caching strategies for layered video [24].

The more static the content, the more the benefit from placing a cache/proxy closer to a client. Such caching methods are unsuitable when data is generated in real time (e.g., games, dynamic content, remote control operations or financial data updates).

Kroeger *et al.* [26] investigated the limits of latency reduction for web caching, and observed only an average 22%–26% reduction, even for cache hit ratios of 47%–52%. This implies that larger objects were less frequently cacheable than small ones.

A *reverse proxy cache* (see Fig. 4(a)) is a popular form of proxy that is typically located close to a content provider's servers. This can reduce response times by holding replicas of a specific subset of the server's content. A reverse proxy cache may also be implemented in an ISP's network to handle requests to popular content providers, adding a small processing delay to offer the benefits of a "normal" proxy cache.

A *load balancer* (see Fig. 4(c)) is an alternative to a reverse proxy. It redirects incoming requests to a server farm to one of a pool of servers that hold either a subset or active replicas of the content. Proactive replication in a server pool can better spread server load and keep response times low; a classical technique is Dynamic Segment Replication (DSR [27]). An example is the Google web query architecture [28], where the www.google.com address is first mapped using DNS to a server farm, where a front-end web-server balances the load to index servers, document servers, spell checkers, ad servers, etc. This requires local processing and data-centre communication (see later) to distribute and forward operation requests, each adding small amounts of delay before a response reaches the user.

Push caching is a type of caching with active replication. A content provider uses information about usage and regional interest to send replicated objects to specific remote caches to optimize response and finishing times.

Methods such as push caching and active replication have been largely replaced by *Content Delivery Networks* (CDNs) (see Fig. 4(b)). A CDN can manage replication and distribution across multiple content providers, providing a multiplexing gain that reduces hardware costs (e.g., the sophisticated edge caching network used for YouTube content and live video [29]).

Measurements by Chen *et al.* [30] distinguished and compared the times to retrieve the small static and dynamic elements of Bing and Google searches from their CDN's frontend and backend nodes. They provided a variety of visualizations of their results, including one that showed that most clients with a small RTT to a frontend node benefit from the CDN (halving retrieval times and better), while clients with larger RTTs benefit mostly from reduced jitter.

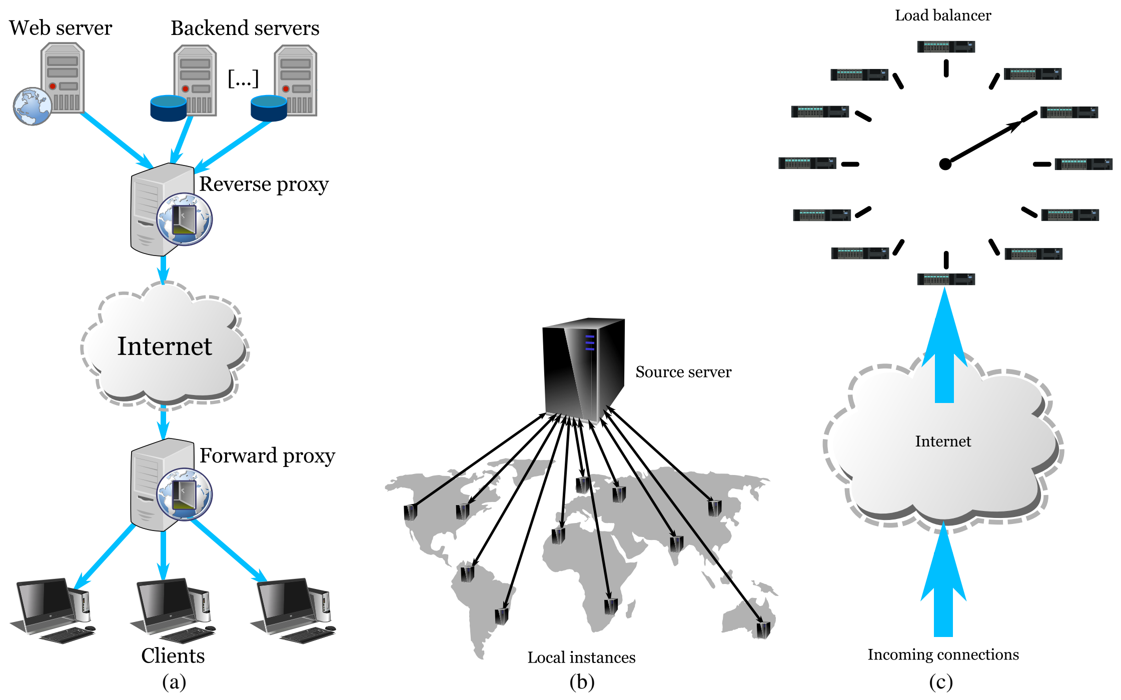


Fig. 4. Examples of solutions for content placement to reduce latency and improve scalability. (a) Placement of forward and reverse proxy. (b) Content delivery network. (c) Load balancing.

Caching approaches are also used for streaming services, and with the current trend of using adaptive segmented HTTP streaming for progressive download of a sequence of video segments, caching infrastructures have become an important means of latency reduction. Creating a streaming experience requires both short finishing times and stable response times for the download of video segments. YouTube has received the most attention of all CDN services. In 2008, Saxena *et al.* [31] distinguished between videos delivered from YouTube servers and YouTube videos delivered from Google's CDN (YouTube has since been integrated into Google). This found that nearly 50% of segments received from the Google CDN had a sub-second finishing time, while YouTube's own servers delivered only 5% of segments with a finishing time below 1 s. By 2012, Google's CDN served all YouTube traffic, using a three-layer caching system that supports conditional caching with a 90% hit rate.

Network proxies may also perform operations and services on behalf of an origin server for other applications. In interactive multi-user applications, collaborative applications or networked virtual environments, well-chosen placement of proxies can reduce the average latency, by migrating application state to a dynamically selected server depending on location and time of the day [32].

2) Client Caches: A local cache at a client can also be used, e.g., Web client caching can reduce latency for repeated requests for the same object. In contrast, data pre-fetching in HTML5 allows a web application to control the set of objects that are cached and allows a user to interact with cached content when offline.

The responsiveness of many modern web applications that execute on a client may be increased by enabling them to automatically receive ("pushed") content [20], [33]. This requires a server to maintain a (persistent) channel to send data, rather than using a separate transport connection for each object. In many cases pushed content can reduce access latency, but when the content competes for limited network resources (e.g., over capacity-limited mobile/wireless links), it can add to network queuing and induce latency for other flows.

3) Prediction and Latency-Hiding: If an application has data transfers that follow strict rules, like physical laws or predictable patterns, it may be able to ask for data to be delivered before the time when the data is actually needed.

Predicting the need for application data relies on recognizing specific application patterns, such as user behavior in an on-line game. When the prediction is correct, such an approach may save the time needed to request and deliver the data (1 RTT + delivery completion time). A cost incurred by such schemes is unnecessary data transfers caused by failed predictions. In many cases, it may be difficult to provide predictions of a quality that is good enough for practical use [34].

Another kind of commonly used prediction, is to hide network latency by enabling the client to predict the expected continued interaction behavior and display the prediction to the user, e.g., dead-reckoning in gaming [35], [36]. Client-side latency hiding does not reduce the actual latency, but can greatly increase the quality of experience for the user. A high degree of wrong predictions may however reduce the experienced quality by inducing extra jitter.

A latency-reducing technique is also known from streaming of virtual scenes. While view interpolation on the receiver side is the essence of image based rendering [37], it does usually not involve latency hiding. Model-based view-extrapolation [38], [39] is a variant that avoids interaction delay from interactive camera movement by interpolating the currently viewed scene of a virtual world based on views that were previously received from a server. The server sends difference images to correct these estimates in retrospect and suppress drift.

D. Service Architecture

The client-server model is a frequently used architecture for distributed systems. A single server endpoint can be replaced by a (dedicated or shared) **server cluster** or a **server farm**. Replication of itself does not improve latency, it only increases capacity, because latency is solely influenced by the distance to the server. With replication, servers are still placed at a fixed (static) location and all client communication is still with one of the servers. Users in different geographical locations will experience different latencies for the same service, influencing fairness, e.g., of an online game or a trading market. The following sections outline various ways to re-arrange the basic replicated server architecture to improve latency.

1) Structured Peer-to-Peer: The Peer-to-Peer (P2P) model has become popular because of its scalability and because it facilitates moving costs from the provider to an ISP and the user, thus reducing the deployment cost for the service provider, but at a higher overall cost for everyone involved due to the overhead of setup and management of the P2P topology. While unstructured P2P networks can be encumbered by large overall latency, structured P2P networks can avoid much of this delay.

A system developed by Kumar *et al.* [40], for example, reduced access latency to 25% of the well-known Chord P2P system by optimizing the overlay structure. Small *et al.* [41] proposed construction of a structured overlay topology to minimize the average global latency.

One major issue with the P2P model is that it can present users with a large latency variance if the selection of peers and tree topology is not carefully chosen. Wisely choosing central "super-peers" and optimizing the construction of the overlay can significantly influence the path latency [42]. The tradeoff for optimizing the topology is that time must be spent calculating optimal topologies, increasing the time it takes before a topology for lower latency is established. Also, the forwarding latency for P2P networks is often higher than when using network-layer routing, because application layer processing is not so likely to be optimized for forwarding. The overhead of management and control in P2P networks tends to increase complexity relative to a central control server with global knowledge. Hybrid client-server/P2P solutions can ease management of such scenarios and help maintain more stable routes for application forwarding where delay variance must be kept low. Standardization by the IETF ALTO working group aims for increased efficiency of cooperative networks with P2P overlays [43].

2) Cloud Server Placement: Cloud-based service architectures move the backend/infrastructure to reside with a provider of Infrastructure as a Service (IaaS), as offered by Amazon Cloud [44], Microsoft Azure [45], Google Compute Engine [46], and others. This placement provides the opportunity to migrate the service between data centers around the world, and offers flexibility to dynamically scale server resources up or down to reflect fluctuating application demand.

By dividing user workloads over several servers placed locally, processing delay may be significantly reduced, and by moving the physical location of the server, the transport latency can be dynamically optimized to meet the demands of the average user of a system at any given time [47], [48]. The latency gain from relocating a server has to be traded against the migration latency. Migration latency depends on the propagation delay between the two locations plus the time to complete movement of operational state. This operational state continually changes (as opposed to the software image and configuration state that can be run and configured in a remote location before arrival of the operational state). Stateless services do not require dynamic operational state, eliminating migration delay, because the service can 'make before break.'

Mobile devices are good candidates for needing resource-intensive applications offloaded to the cloud, but high WAN latencies (and/or relatively low link capacities) may hinder this. Cloudlets [49] push the idea of datacentre relocation further by placing resources very close to wireless terminals to facilitate, for instance, cognitive-augmentation applications such as natural language translation or face recognition.

A downside of IaaS is that it may lead to larger variance in processing time due to time-slicing of the hardware resources. If an application has strict processing deadlines, the need for direct hardware control and predictable processing time may outweigh the flexibility of the IaaS model. Still, research on using IaaS for real-time services indicates that the solutions deployed today are able to meet the requirements for some real-time services [50], [51].

3) Cloud Cache Placement: Another approach is to allow caches (see Section II-C) to be spawned dynamically at a location close to a user group. Since IaaS allows resources to be scaled to meet the demands of an application, optimizations may be implemented to dynamically adjust the content needed and cache size to adapt to the local needs.

Globe-spanning Cloud services like Amazon EC2 can now compete with P2P services for many latency-sensitive applications, since they reduce costs for service providers, offer scalability and provide low response times. However, using the clients as intermediate nodes for a hybrid client-server/P2P model may yield latency benefits when combined with caching and prediction mechanisms (see Section II-C).

4) Virtualizing Chains of Network Functions: In certain scenarios it is common for traffic to be passed through numerous network functions, e.g., each wide-area link in an enterprise may include a firewall, an intrusion detection system, a WAN accelerator, a voice-gateway, an application-layer-proxy and an edge-router; mobile networks contain long function chains too.

If each function is deployed as a separate physical box (Fig. 5(a)), in the worst case the data would have to be serialized or de-serialized four times per function—out of an intermediate switch, into and out of the function and back into the switch. All serialization delay can be removed by executing the whole chain as virtualized network functions in the memory address space of a single general-purpose server (Fig. 5(b)). Comparison of the block diagrams on the right of the figures illustrates this point. If modern virtualization techniques such as single root I/O virtualization (SR-IOV [52]) are used, processing

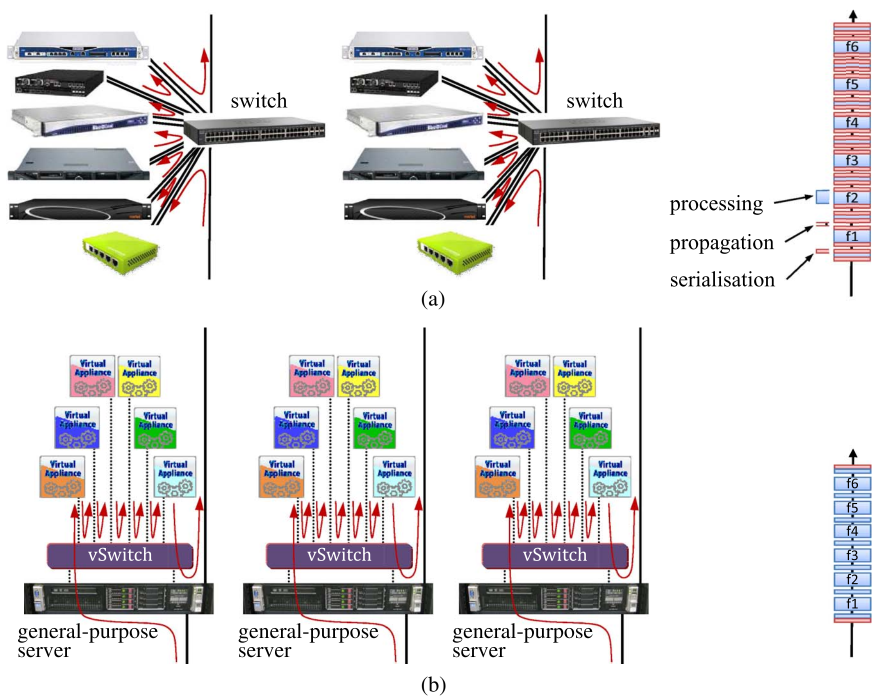


Fig. 5. Removal of repeated serialization delay using network functions virtualization. (a) A workload divided over 2 chains of 6 physical network appliances. (b) The same workload divided over 3 general-purpose servers running virtualized chains of the same functions.

and switching latency (see Section IV-E) can match dedicated hardware. And, in the virtualized case, bits can move between functions in parallel over the memory bus, rather than in series over network links.

Note that, in the hardware case, cut-through switching can bypass the two serialization delays into and out of the switch, at least for the payload (see Section IV-C). However, most of the functions chained together in the above enterprise example act on the payload, so cut-through cannot bypass any of the serialization delay into or out of these functions. Also note that serialization delay is typically small (e.g., 12 μs for a 1500 B packet at 1 Gb/s) relative to wide-area propagation delays of milliseconds. So the sub-millisecond improvement from saving even repeated serializations may only be significant for short paths.

Fig. 5 illustrates how a) physical appliances dedicate computing resources to each function, whereas b) the virtualized approach dedicates computing resources to subsets of the users' workload. Thus, if one server is not powerful enough to execute the whole chain of virtualized functions, the workload (not the chain) can be divided over more servers. In the original physical case, the workload is often divided over at least two chains for resilience anyway (as shown in Fig. 5(a)).

III. Interaction Between Endpoints

This section examines the latency introduced by end-to-end protocols. Transport protocols operate over the network path between a pair of transport endpoints. These transports may be datagram-based (e.g., UDP) or connection-oriented (e.g., TCP). These protocols can incur multiple control interactions between the endpoints before data communication may start. End-to-end protocol setup can also be needed at high layers (e.g., for security).

TABLE I
Number of RTTs Required Before Data Transmission Can Start (Adapted From [54, Table 1]). Resumed Sessions Over TCP Connections are Shown Without and With TCP Fast Open (TFO)

	Number of RTTs		
Protocol	**First connection**	**Resumed session**	**Resumed session with TFO [55]**
IPsec [56]	≥ 3	1	n/a
TCP [57]	1	1	0
TCP then TLS [53]	3	2 or 2.5	1 or 1.5
TCP then TLS False Start [58]	2	2 or 1.5	1 or 0.5
TCP then TLS Snap Start [59]	2 (poss. 1)	1	0
Tcpcrypt [60]	2	1	TBA
MinimaLT [54]	0	0	n/a

An end-to-end protocol can also trigger control interactions during a session, for instance to recover lost packets to provide reliable transfer, or to assess the characteristics of a path (e.g., available capacity, support for network features). Each interaction incurs at least 1 RTT to communicate over the path to the remote endpoint and receive a response. Reducing such interactions can have a dramatic impact on latency, especially for short data transactions.

Finally, an end-to-end protocol may merge messages (either carrying application data, or only protocol signals) for optimization purposes. Such optimizations, which reduce the number of packets sent on the wire, may in some cases result in additional latency.

Table I shows the number of RTTs required by certain protocols before data can be transferred. If the host is unknown, a DNS lookup (≤ 1 RTT to a remote resolver) is first required (not shown in the table). If the DNS information is cached, this RTT is avoided in future connection attempts. The column marked 'Resumed session' shows how some protocols cache connection state to speed-up subsequent communications that use the same protocol to connect to the same host. Note that the second alternative figure shown in Table I against Transport Layer Security (TLS v1.2 [53]) is for the case where the server, not the client, resumes sending data first. This makes one flight (half a round trip) difference as illustrated in Fig. 9.

The remainder of the section examines how endpoint interaction latency can be reduced by improvements to the initialization of the transport mechanisms, security sessions or middlebox traversal, by choice of end-to-end packet loss recovery mechanisms, or by improving (or disabling altogether) message-aggregation mechanisms.

A. Transport Initialization

Startup latency arises from the time to complete a protocol handshake (e.g., at the start of a TCP [57], SCTP [61] or DCCP [62] connection). At least one RTT of delay is incurred for each sequentially completed handshake. In practice this delay can be larger when there is loss (due to congestion, or to unsupported features in the network and/or a middlebox), requiring a timeout to restart the exchange. Communication options, even latency reducing options, may need to be negotiated at the start of the session. These can include: selection of alternate transport protocols, features such as Explicit Congestion Notification (ECN [63]), use of relays to overcome the limitations of middleboxes, etc. Each feature that requires negotiation can potentially add further startup latency.

The impact of startup latency can be mitigated by reducing the number of sequential protocol exchanges, and by multiplexing data over an existing session or by persistent use of a session (rather than opening and closing a session for each transfer).

1) Parallel Option Negotiation: Parallelizing option negotiation can significantly reduce protocol initialization latency. Happy Eyeballs (RFC 6555 [64]) allows simultaneous attempts to connect using IPv4 and IPv6. This appears to double the number of connection attempts, but by using a caching strategy to store previous successful connectivity, the algorithm can minimize the number of attempts and offer substantial delay improvement. Most modern web browsers, i.e., Chrome, Firefox, Safari, and Opera, support this algorithm.

This idea could be extended to other communication options. For instance, many middleboxes inadvertently block the Stream Control Transmission Protocol (SCTP) from end-users even though it is usefully employed within the interior of networks. Therefore, it would be desirable for end-users to try to use SCTP if available, otherwise to fall back to TCP. Reference [65] describes this approach, which is illustrated in Fig. 6; the client attempts to simultaneously open both a TCP connection and an SCTP association and, if the latter succeeds within a short time interval, the former is aborted then the application continues using SCTP. This may allow the application to benefit from SCTP's latency reduction mechanisms, e.g., reliable out-of-order delivery without head-of-line (HOL) latency (Section VI-B).

2) Reducing NAT Setup Delay: Many home networks employ port-mapping Network Address Port Translation (NAPT) to share use of a single assigned address. A NAPT performs

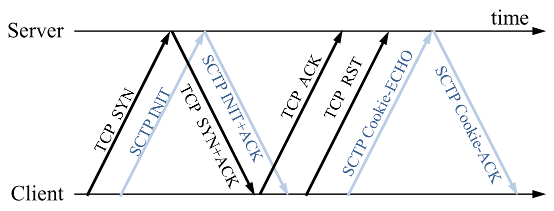


Fig. 6. Happy Eyeballs technique applied to the setting up of transport-layer connections.

mapping of addresses and rewriting of packet headers during router forwarding. For data download to a device behind a NAPT, this has little impact on latency, but for communication initiated by a device on the public side of the NAPT (e.g., a VoIP or teleconference media call), this can incur latency communicating with an off-path server(s) to discover a viable candidate path between the endpoints, e.g., Session Initiation Protocol (SIP [66]), signaling to invoke Interactive Connection Establishment (ICE [67]). Once potential candidates are discovered, connectivity checks are required to determine that a candidate path is viable. This can result in appreciable handshaking delay [68]. The delay can be mitigated by starting ICE connectivity checks before the signaling completes negotiation to set up the media flow, this parallel discovery can reduce the overall latency for a flow to start. Some delays can also be mitigated using P2PSIP, instead of a central SIP server [69]. When multiple candidate paths are found, ICE may be used to minimize latency on the media path (this can incur additional delay, waiting for more potential candidates to be validated).

In some cases the resultant path requires an off-path relay using an application layer intermediary, e.g., Traversal Using Relays around NAT (TURN [70]). A TURN media relay causes packets to follow a longer than necessary path and incurs additional latency in the media/data path [68], [69]. Well-placed media relays (e.g., at network borders) can significantly reduce the overhead of routing over a longer path. TURN may also be used to impose a policy, e.g., in a corporate environment.

3) Fast Opening of TCP Connections: The current TCP standard requires a TCP connection to start with a three-way handshake that adds a round trip of delay before the server can receive any request data.

The traditional three-way handshake serves multiple purposes, not all of which were originally intended:

1) it allows a server to test that the client really is at the address it claims to be using;
2) it synchronizes use of sequence numbers;
3) it has come to be used by middleboxes (including NATs and firewalls) to initiate storage of connection state;
4) it allows the two endpoints to agree on the use of protocol options;
5) it allows both endpoints to estimate the current round trip time before sending data;
6) it allows a connection to detect congestive loss in each direction using a minimal size packet;
7) it allows the client to shut down any duplicate connections the server unwittingly opens in cases when a client's opening packet was duplicated.

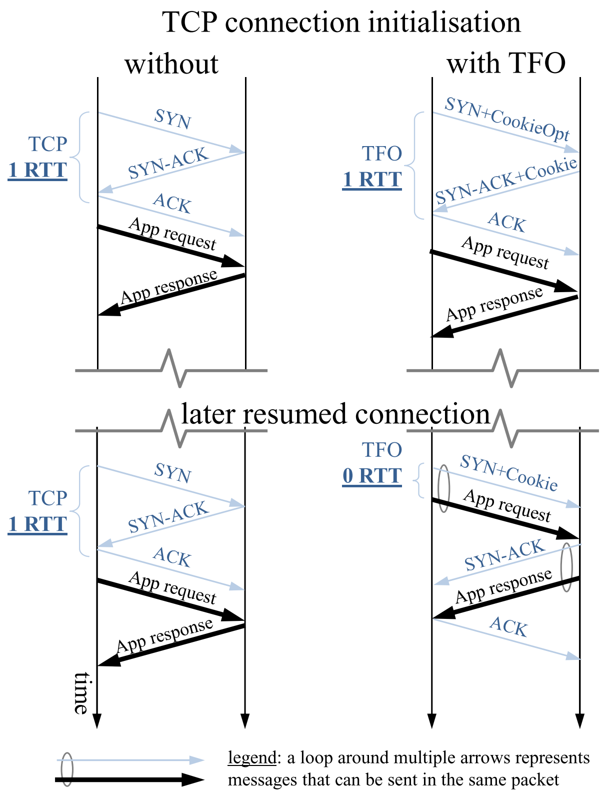


Fig. 7. TCP Fast Open (TFO) saves a round trip when resuming a connection.

Transaction TCP (T/TCP [71]) was an early attempt to reduce latency for short transactional connections by replacing TCP's three-way handshake, instead storing a counter per client at the server. However, once various security flaws were discovered [72], T/TCP was not pursued further and the IETF eventually moved its status to Historic [73].

TCP Fast Open (TFO [55]) is a more recent attempt to allow a TCP connection to circumvent the three-way handshake when the client opens a connection to a server to which it has recently connected. The server does not have to hold any state between connections, which simplifies load balancing in large server farms. Instead, the server gives the client a TFO cookie that the client sends when it opens a subsequent connection. This allows a server to verify that an earlier valid handshake had been completed. A client that sends an opening (SYN) packet with a TFO cookie can include application data in that packet, which the receiving endpoint can pass straight to the application (represented by the loop[1] in Fig. 7 joining the SYN with the application request). In most cases this will reduce latency, but it could incur additional delay, e.g., if a middlebox blocks the cookie option.

A TFO client's SYN packet and TFO cookie serves the first four of the above purposes (except it may fail to achieve the first behind a NAT because then it cannot detect client address spoofing), but not the last three. Therefore, when TFO resumes a connection with an initial window of data, if the data is not received, it will not have a current estimate of how long it is safe to wait before retrying (item 5), and it will be too late to discover that it should have sent less (item 6). The lack of duplicate connection detection (item 7) means that TFO is not a universal replacement for standard TCP, and must only be used by applications that either test for duplicate connections themselves or inherently do not care about duplicate connections (e.g., pure look-up semantics). It is well-suited to many latency-sensitive applications, such as web applications. Radhakrishnan *et al.* [74] provides further rationale for TFO, and evaluates its merits against alternative approaches.

Accelerated Secure Association Protocol (ASAP [75]) is a proposal with goals similar to TFO—to eliminate the RTT needed to complete the TCP three-way handshake. In contrast to TFO, ASAP piggybacks transport-connection establishment on a DNS lookup request. This requires modifications to the DNS (namely, to the authoritative name server).

4) Application Pipelining: The earlier versions of the Web protocols required unnecessary set-up of connections, with consequent delays, but they have evolved to meet changing demands and optimize performance. Web 2.0 has significantly altered the characteristics of in-line objects, by allowing embedding of additional services, e.g., for audio/video streaming, or interaction with an Online Social Network (OSN). HTTP 1.0 [76] relies on multiple connections for concurrency. This can add latency due to connection setup/teardown and TCP slow-start. HTTP 1.1 [77] addressed this by allowing persistent connections and pipelining of HTTP requests. This enabled a client to decide to pipeline requests, which reduces the delay waiting to request, but could lead to "Head-Of-Line" (HOL) blocking delay (see Section VI-B) when a single transport is used, e.g., when the first in a series of requests includes dynamically generated content.

SPDY [78] introduces a number of latency-reducing techniques, including a session layer that builds on pipelining in HTTP 1.1 by supporting multiplexing of *concurrent* streams over a single persistent TCP connection. SPDY is described more fully in Section VII-A where protocols that combine numerous techniques are outlined.

5) Path MTU Discovery: Although most Internet links support a Maximum Transmission Unit (MTU) of at least 1500 B, many paths cannot sustain a Path MTU (PMTU) of this size, due to one or more links with a lower MTU. Although a lower MTU can reduce head-of-line latency from other flows sharing a slow link, the more common reason for a reduced MTU is the use of tunnel encapsulations. Path MTU discovery is a challenge when using such paths. The original path MTU discovery (PMTUD) algorithm [79] had drawbacks that could introduce intermittent extra delay or even prevent communication [80]. This was updated in the more robust packetization layer PMTUD [81], but this still relies on occasional probe packets sent along the path, which can introduce recovery delay when a probe is larger than the actual PMTU supported, and hence needs to be retransmitted. A discussion of PMTUD and issues relating to use with tunnels is given in [82].

[1]In the case without TFO, an application can tell TCP to combine the ACK ending the 3-way handshake with the application request. By default TCP sends the ACK separately, so we have not shown a loop here.

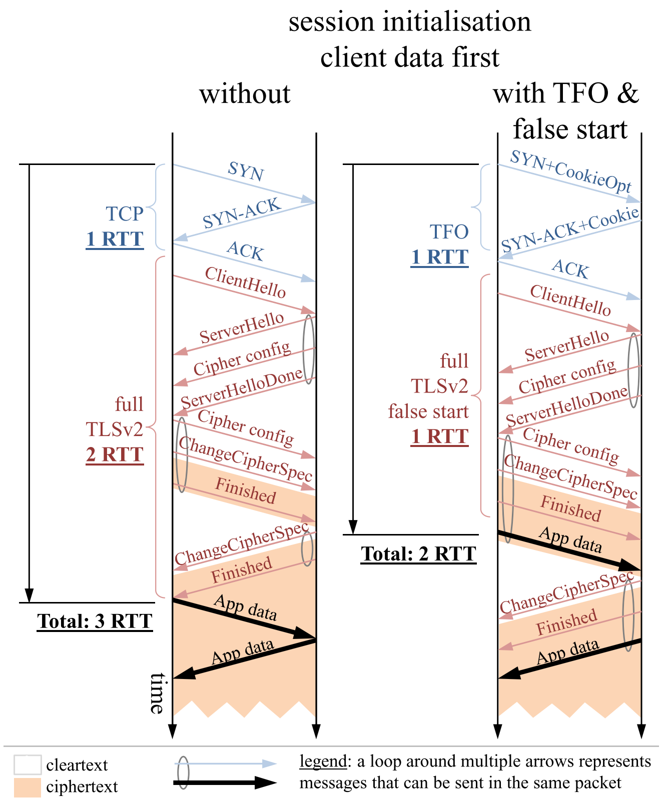


Fig. 8. Protocol timing diagram showing handshaking to initialize a transport layer security session. False Start saves one round trip, but at this initial stage TCP Fast Open (TFO) saves nothing, although it sets a cookie that may save time later.

B. Secure Session Initialization

Security mechanisms have become the norm in Internet communication. The use of an unmodified security protocol can add significant latency to setup of an application flow as illustrated in Table I. This has prompted proposals to update security protocol interactions to reduce the number of RTTs, with the potential to provide significant latency gains for short sessions.

1) Faster Transport Security Negotiation: Transport Layer Security (TLS) is the IETF-standardized successor to the Secure Sockets Layer (SSL) that is universally used to authenticate and encrypt HTTP connections (denoted by the 'https:' URL prefix). In TLS up to v1.2 [53] the handshake adds 2 RTTs to the set-up of a typical connection where the client sends data first, e.g., HTTP. Here we focus on TLS over connection-oriented transports (TCP, SCTP, etc.), but similar approaches could apply to datagram TLS (DTLS [83]).

a) TLS false start: False Start [58] begins sending application data 1 RTT earlier than typical for TLS (see Table I). The initial TLS handshake messages cannot be encrypted themselves because they negotiate encryption keys and certificates. Instead, the respective 'Finished' messages that the client and server use to complete the TLS handshake both include a hash to validate all the previous messages in the handshake. False Start allows a client to start the stream of encrypted application data directly after its own 'Finished' message (Fig. 8), whereas, in the original TLS, a client waits 1 RTT to receive the corresponding 'Finished' message from the server before transmitting encrypted data.

If a client resumes an earlier session, the original TLS protocol already permits an abbreviated handshake that takes only 3 sets of messages (half a round trip each) instead of 4. A client cannot use the False Start approach to improve on this, because in the abbreviated handshake a regular TLS client already starts transmitting encrypted data directly after its 'Finished' message (see Fig. 9(a)), because, in contrast to the full handshake, the server sends 'Finished' first.

On the other hand, if the server resumes the session first, e.g., it has updates to a previous response, it can use False Start to save 1 RTT (see Fig. 9(b)). This is why Table I shows two alternative delays for a resumed False Start session. Fig. 9 also shows how TCP Fast Open (TFO—see Section III-A3) can be combined with TLS False Start to save an additional RTT.

Google deployed False Start in Autumn 2010, on the assumption that a unilateral change at the client end would work with all existing servers. However, an unacceptably large number of SSL terminator hardware middleboxes occasionally and non-deterministically failed to support the protocol, probably because of unexpected message timing. Therefore, since April 2012 False Start has been disabled in Chrome with one important exception; when a server confirms support for Next Protocol Negotiation (NPN [84], [85]) or Application Layer Protocol Negotiation (ALPN [86]). A server farm that supports NPN or ALPN is assumed to have upgraded any SSL termination middleboxes. Use of NPN and ALPN allows a session on a HTTPS port to use any application protocol, not just HTTP, without additional rounds of negotiation. This enables new protocols such as SPDY (see Section VII-A) and HTTP/2 to be used through middleboxes that block ports other than HTTP and HTTPS. And with False Start these protocols can be secured in one handshaking round, not two.

b) TLS Snap Start: TLS Snap Start [59] enables a client to send encrypted and authenticated application data without waiting for any handshake from the server. The client needs certain information about the server (its cipher suites, its public certificate, etc.). Therefore Snap Start is not applicable for ephemeral key techniques (e.g., Diffie-Hellman), nor for sessions resumed by the server end.

Because the pre-requisite server information is public and fairly stable, it can have been retrieved from a directory, e.g., using the Session Keys protocol [87], or pre-fetched when opening a referring Web page [88]; the client does not need a prior session with the server itself. Therefore, Snap Start is more generally applicable than just session resumption. Resuming a session could never be better than Snap Start, because it creates new secrets for each new session without any handshaking delay. It also does not need to store secrets between sessions.

The insight of Snap Start is that the random information that the server provides for TLS does not need to be random; it only needs to be unique to prevent replay of the whole session. Therefore, the client suggests a unique value for 'server random' and timestamps it. It then sends a hash of the whole handshake it predicts the server would have used. Without waiting, it then optimistically encrypts its first message to the server

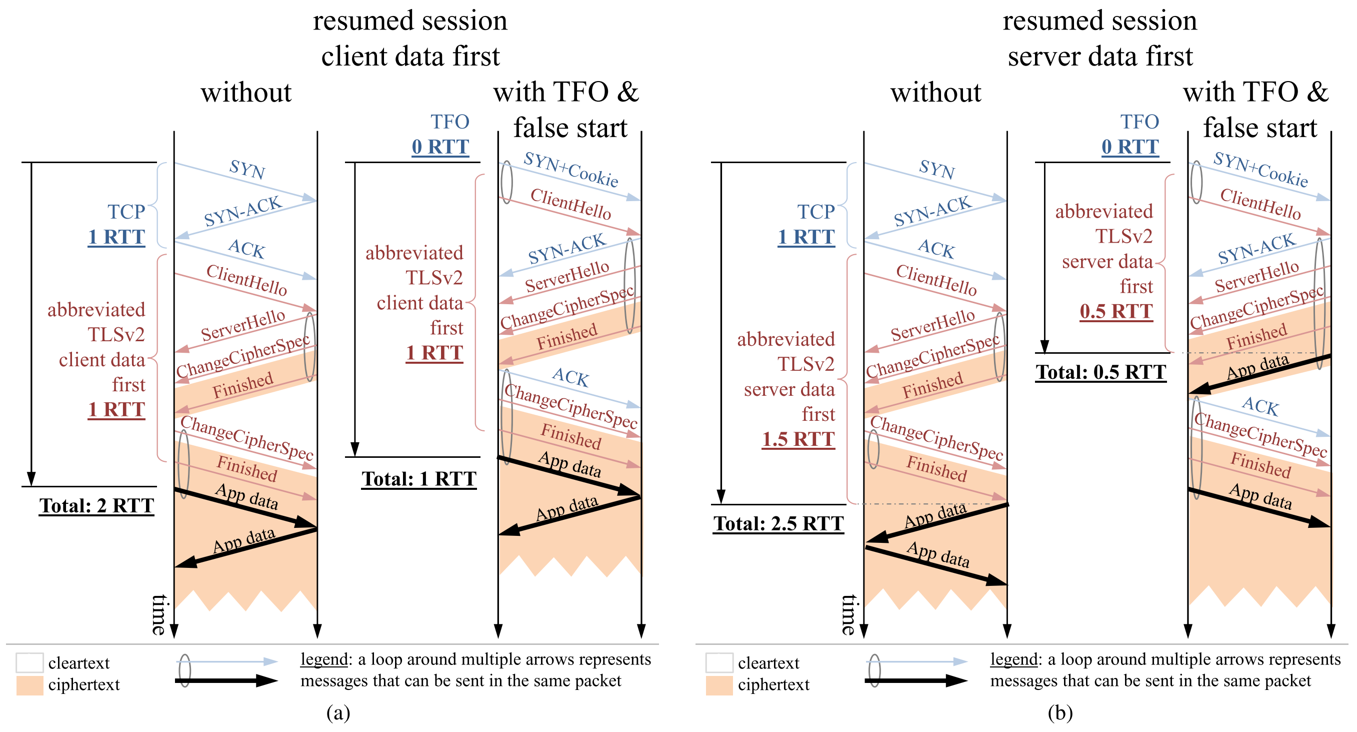


Fig. 9. Protocol timing diagram showing handshaking to resume a transport layer security session. (a) Client data sent first. False Start saves no time in this case, but TFO saves one round trip. (b) Server data sent first. Both False Start and TFO save one round trip each in this case.

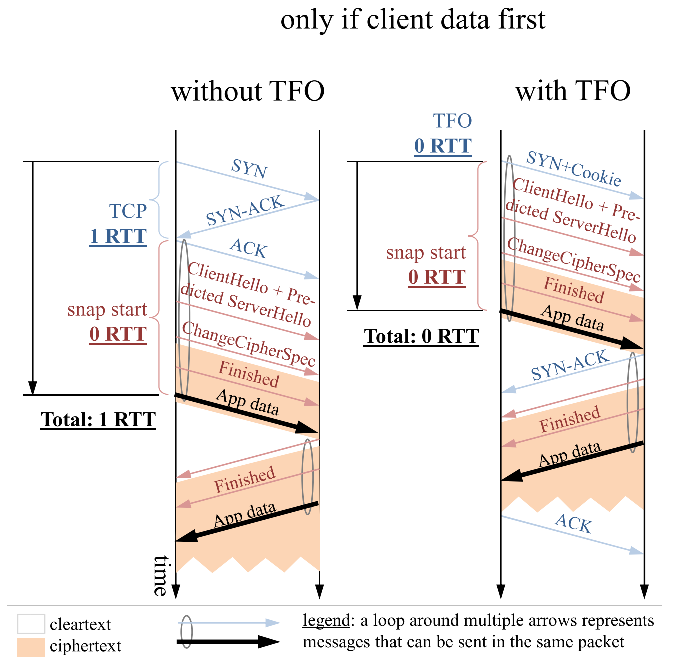


Fig. 10. TLS Snap Start timing diagram, showing it can remove all handshaking delay from TLS.

using these cryptographic choices (see Fig. 10). The server only proceeds if its own hash of the cryptographic choices matches the client's prediction and the value of 'server random' is indeed unique. Uniqueness is checked against the 'strike-list' of values used by clients. To limit storage and look-up overhead, a server can limit the oldest allowed timestamp (so it could reject clients that are not at least loosely synchronized). The allowed range of random numbers may be limited by publishing a well-known 'orbit' value that clients must look up along with the other public information.

Snap Start was developed by Google, and deployed on their servers and in the Chrome browser in 2010. However on 23rd June 2011 Google decided to withdraw it, for reasons that remain unclear. The method exhibits an inherent vulnerability to downgrade attacks if any cipher suite that was previously considered secure is found to be compromised. However, that risk was known before Snap Start was released. A more plausible reason for withdrawal is the method's operational complexity. Nonetheless, the promise held out by Snap Start inspired the IETF to start work on a new low latency version of TLS (v1.3) in late 2013 [89]. The focus is on new ways to ensure session uniqueness against replay attacks.

2) Building Encryption Into TCP: TCPcrypt [60] is a proposal to allow a client to negotiate encryption for every connection, but fall-back to traditional TCP if a server does not support this. TCPcrypt negotiation uses a TCP option and requires 1 extra RTT to exchange cipher material (in a separate channel to the TCP data) (see Table I).

A new TCP connection can resume a TCPcrypt session in 1 RTT, by exchanging new keys piggy-backed on the initial exchange in the TCP SYN packets. TCPcrypt has not (yet) been designed to reduce latency further when resuming a session, by carrying data encrypted with the new keys on the SYN packet.

3) Bootstrapping Security From the DNS: Minimal latency networking (MinimaLT [54]) provides an example of a 'clean-slate' approach that requires modifications to the client, the server and the DNS. The data server arranges for the DNS to provide an ephemeral public key when a client resolves its name in the DNS. The need to regularly update the server's ephemeral public keys requires the addition of an ephemeral key upload service to the DNS. Petullo *et al.* [54] describe how clients and servers would initiate the necessary long-standing secure connections with the directory service and the ephemeral key upload service. This change has the benefit of 0 RTTs for a client to establish a secure session with a data server. However, even the Domain Name System Security Extensions (DNSSEC) have proven hard to deploy. Therefore choosing to require a different security framework around the DNS may present a tough deployment challenge.

A Snap Start client (Section III-B1b) could obtain its prior knowledge of the server's cryptographic details from the DNS, but Snap Start is less prescriptive about precisely how to get this pre-requisite information.

C. Packet Loss Recovery Delays

A range of Internet transport services have been defined. Many transports such as TCP and SCTP offer an end-to-end reliable service. Other applications choose to use a UDP or UDP-Lite [90] transport, and then implement loss recovery methods in the application itself. Another group of applications require some form of congestion control, but seek to bound latency. Such applications may utilize Datagram Congestion Control Protocol (DCCP [62]) to provide a range of pluggable congestion-control algorithms (but DCCP has proved hard to deploy).

Transport-layer error/loss control can be a source of latency, especially when the data traverses a link with appreciable packet loss/corruption due to link errors and/or a heavily loaded network bottleneck suffering congestion. Measurements reported in Flach *et al.* [91] show that Web flows experiencing loss can see a fivefold increase in average completion time, making loss recovery delays a dominating factor for Web latency.

Three methods may be used to recover a loss: retransmission, redundancy and loss concealment.

A retransmission method uses a control loop that detects loss/corruption and retransmits all missing packets. When packet ordering needs to be preserved, this also implies head-of-line blocking at the receiver to reorder the retransmitted data (see Section VI-B). When the retransmission fails for any reason (e.g., subsequent further loss), it often requires a retransmission timer to trigger (incurring further delay). Retransmission may be implemented at the link (e.g., in a wireless or modem link driver) and/or at the transport layer (e.g., within TCP or SCTP). Implementing link layer retransmission can reduce latency (recovery may be faster), but may also result in more jitter to time-sensitive flows, which may not even need reliable delivery. Also poorly designed methods could incur unnecessary retransmission by the transport protocol [92].

Redundancy may be implemented as simple packet duplication at the sender (e.g., sending multiple copies of a single control packet) or by coding a combination of packets using Forward Error Correction (FEC). FEC enables a trade-off between decreased capacity and enhanced reliability, and additional processing at the sender and receiver. FEC encoding and decoding is typically applied to blocks of data, that can incur latency. On a link, correction codes can achieve a statistical guarantee of reliability with a bounded additional processing delay. In contrast, FEC at the network and transport layers (e.g., transport packet FEC is widely used for high quality video [93]) usually uses erasure codes to encode groups of packets.

A combination of the three loss recovery methods, further outlined below, may be required to achieve a tradeoff between processing, reliability and delay.

1) Application Tolerance to Loss: The set of applications that require all delivered data to be uncorrupted, but do not require loss-free transmission or ordered delivery can use the services offered by UDP and DCCP. This can save significant time when loss is experienced by removing the need to reliably detect loss, retransmit packets, and reorder the data (Section VI-B).

Transport protocols that enable partial reliability can allow an application to conceal rather than retransmit/correct any network loss/corruption, e.g., prediction of missing video content, or suppression of corrupted voice samples. Stewart *et al.* [94] provide a full framework for a partially reliable service in SCTP, giving a *timed reliability* service as an example. Methods have also been proposed as enhancements to TCP. Mukherjee and Brecht [95] introduced the concept of data having a useful life time with Time-Lined TCP, allowing reliability to be traded for reduced latency. McCreary *et al.* [96] exploit application tolerance to loss (and not delay) using a receiver-only TCP modification.

Where a network path is error-tolerant (e.g., some radio links), the UDP-Lite or DCCP transports can be configured to allow the application to appropriately handle errors in the transport bit stream. Concealing errors can eliminate additional delay.

2) Reduce Packet Loss Detection Times: There are a number of proposals that seek to detect loss earlier for reliable transports. Hurtig *et al.* [97] recommend updating the TCP and SCTP retransmission timeout (RTO) timer to reduce the latency from detecting loss with short or application-limited flows. Early Retransmit [98] modifies TCP and SCTP when the congestion window is small to more quickly decide that a packet has been lost, thus reducing the delay associated with a timeout. The basic idea of early retransmit (Fig. 11), is to allow the TCP Fast Retransmit to be triggered by a number of duplicate acknowledgements smaller than the standard value (three). Mellia *et al.* [99] seek to avoid TCP RTO expiry by carefully segmenting TCP data, to try to always have at least three segments in-flight, enabling Fast Recovery. Dukkipati *et al.* [100] and Flach *et al.* [91] propose Tail Loss Probe, a modification to TCP to avoid delay when the end of a packet burst is lost. Instead of waiting for a RTO, the last packet is retransmitted after about 2 RTTs as a probe to trigger Fast Recovery based on SACK/FACK (selective/fast acknowledgement).

3) Combining Redundancy and Retransmission: Redundancy can be added to reliable link and transport protocols to avoid delays due to loss detection and recovery. This technique has been long-used on radio links forming a class of methods known as 'hybrid ARQ' (see Section IV-D for a discussion on link FEC).

Some researchers have also proposed using packet FEC combined with TCP. These approaches trade network capacity

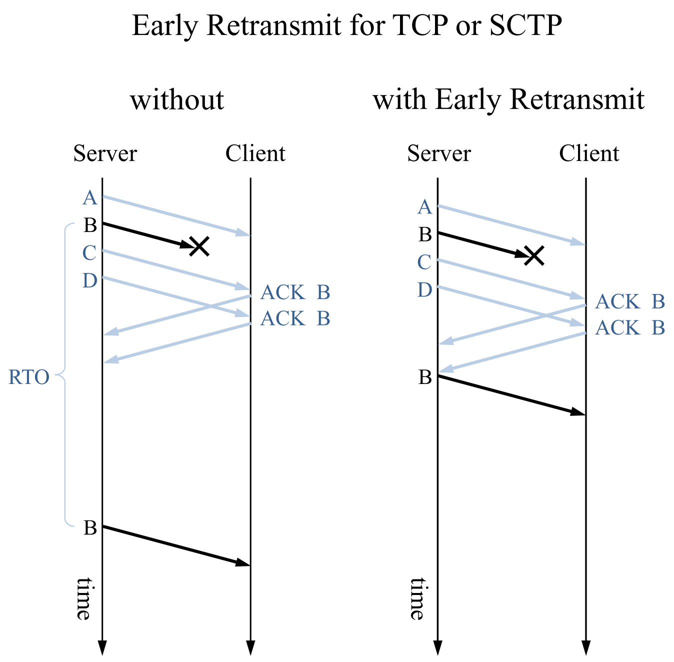


Fig. 11. Early retransmit for TCP or SCTP.

to reduce/eliminate retransmission delay when reliable in-sequence delivery is required.

LT-TCP [101], [102] proposed a sender and receiver update to provide a proactive FEC mechanism based on the end-to-end packet erasure rate to minimize the need for retransmissions, and reactive retransmission with FEC to reduce the risk of TCP timeouts during retransmissions. A similar approach is also suggested in Flach *et al.* [91].

Evensen *et al.* [103] presented Redundant Data Bundling (RDB), with a sender-side-only modification to TCP that retransmits unacknowledged data redundantly with new data as long as the packet size does not exceed the network maximum segment size (MSS). For applications that produce small packets, such as online games, this often avoids the need for retransmissions by timeout, reducing the observed latency upon loss.

4) Explicit Congestion Notification: Loss within a packet network is not only used to remove data from the network to alleviate congestion, but senders deliberately increase their rate to induce loss as a transport signal they can use to regulate their rate. There are also multiple other reasons why a packet might not (yet) have appeared at the receiver, e.g., re-ordering, transmission errors, packet size errors, routing errors, policy violation. Explicit Congestion Notification (ECN [63], [104]) is a method that can allow IP-aware equipment (routers, firewalls, etc.) and other lower-layer devices [105] to propagate an unambiguous congestion signal via a field in the IP packet header. This reduces the wait needed to determine that a gap in a sequence of packets can be considered as a loss, not just re-ordering. It also removes the wait needed for a subsequent retransmission (see Section III-C2).

ECN requires some form of active queue management to detect early build-up of queues. Depending on the equipment configuration, ECN can also help to reduce delay in other ways; these are discussed in Section IV-F6 and Section IV-F7c concerning techniques based on data centre TCP (DCTCP).

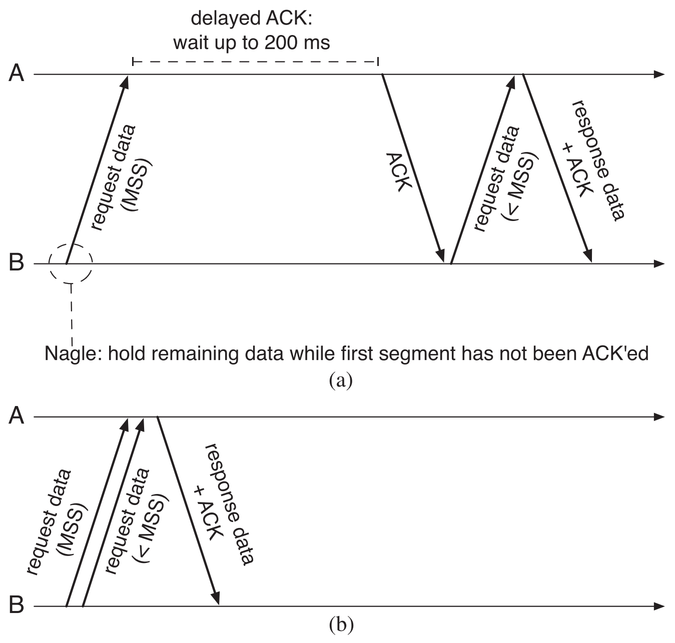


Fig. 12. Combined effect of the Nagle and delayed-ACK algorithms on the latency of a request-response transaction, where the size of the request is over one but under two full-sized segments. (a) Due to the Nagle algorithm, B delays the second segment that completes the request until it receives A's acknowledgement of the first segment of the request. However, A delays this ACK until its delayed-ACK timer fires. Therefore, completion of the exchange is extended by roughly 1 RTT plus the duration of the timer. (b) When B uses modified Nagle [108] or disables Nagle completely, it send both segments of the request without waiting for an ACK, so the exchange only takes roughly 1 RTT.

D. Message Aggregation Delays

Transport protocols such as TCP may aggregate messages (carrying either upper-layer data or protocol signals) to reduce the number of IP packets sent. Thus, the focus of message merging is bandwidth efficiency. Two complementary techniques are commonly used by TCP for this purpose:

- The *Nagle algorithm* [106] tries to limit the amount of small data-carrying packets that are sent (i.e., TCP segments of size $<$ MSS bytes). The algorithm delays the sending of a small segment while a previously-sent segment has not been acknowledged. The goal is to try to coalesce small blocks of application data into a larger block than can be sent in a single packet, instead of sending as many packets as data blocks—i.e., transmission of the first small block is delayed in the hope that the application may produce more data to send while waiting for an ACK.
- The *delayed ACKs algorithm* [107] tries to limit the amount of pure ACK messages (i.e., containing no data), either by piggybacking an ACK signal on a data-carrying segment, or by sending a pure ACK only for every two full-sized data segments. A timer—as high as 200 ms in many systems—ensures that an ACK is always eventually sent even if no data is flowing in the reverse direction.

Both mechanisms trade latency for bandwidth efficiency, but when used in combination they may give rise to severe additional delay [109]. This issue is illustrated in Fig. 12(a), for a request-response transaction where the size of the request is over one but under two full-sized segments.

There are essentially two ways to avoid such latency penalty. First, the Nagle algorithm can be turned off by latency-sensitive applications via a standard socket option (often called `TCP_NODELAY`). Second, the TCP sender can implement a variant [108] of the Nagle algorithm, where transmission of a small segment is delayed *only* if the previously-sent, unacknowledged segment is *also* small. The effect of either of these two fixes is depicted in Fig. 12(b).

IV. Delays Along Transmission Paths

This section examines the delays encountered by a single packet as it travels along the communication path between endpoints, i.e., the *flight* latency from endpoint transmission to endpoint reception. The flight latency is the sum of delays due to: A) Signal propagation delay, B) Medium acquisition delays, C) Serialization delay, D) Link error recovery delays, E) Switching/forwarding delay, and F) Queuing delay. The following subsections detail these sources of delay and survey techniques for reducing them.

A. Signal Propagation Delay

Electromagnetic waves travel about 300 mm per nanosecond in air, slightly slower than the speed of light in vacuum. In guided media the propagation speed is slower, around 200 mm per nanosecond, and slightly slower in fiber than in copper. The propagation delay is linearly proportional to the length of the cable, or, in the case of an unguided transmission medium (like air), the shortest distance between the transmitter and the receiver. The distance that a signal can travel is constrained by the attenuation in the transmission medium. If the distance is too long, an optical signal needs to be amplified or an electrical signal regenerated, incurring delay. An amplifier or regenerator that needs to convert an optical signal to an electrical one, will add significant delay, while an all optical amplifier or regenerator will incur almost no extra delay [110]. Techniques to further reduce propagation delay include making the link path shorter and using a medium which propagates the signal more quickly.

1) Straighter Cable Paths: Current cables tend to follow the easiest path from one location to another, often alongside railway lines or roads, since rights of way are easier to obtain and a path through the land has already been cut. A straighter route would lead to lower propagation delay and potentially fewer signal repeaters.

2) Higher Signal Velocity: While the speed of transmission is fixed for a medium, a hollow core fiber (also known as hollow core photonic band gap fiber or HC-PBGF) may decrease latency and also offer significantly higher bandwidth than conventional monomode fiber. This is typically constructed from a bundle of glass fibers with the central ones missing (Fig. 13). Although the signal propagates by diffraction through the fiber cladding as well as the hollow core, signals propagate at close to the speed of light in air.

There are niche commercial applications using hollow core fiber, however, the loss levels are still unusable for telecommunications. By March 2012, losses of 3.5 dB/km had been achieved [111], down from 13 dB/km in 2003. Further research is required to reach 0.2 dB/km (the performance of conventional fiber). Until this is achievable, overall latency is higher due to the need for repeaters every 7 km instead of every 120 km.

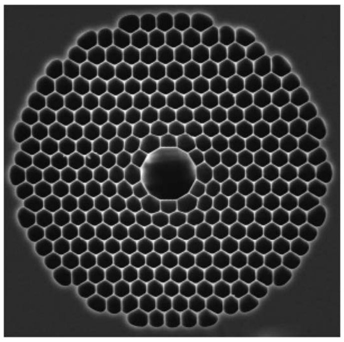


Fig. 13. Cross-section of a hollow photonic crystal fiber (PCF). *Source: NKT Photonics.*

3) Higher Velocity With Straighter Routes: Point-to-point microwave links are increasingly being deployed to transmit data with less delay than fiber, e.g., between financial centers. It is easier to achieve a straighter route with microwave, because planning permission is only required for the towers along the route instead of for laying fiber the entire length of the route. This also enables obstacles such as roads, buildings, lakes and rivers to be overcome with relative ease. However the signal experiences far higher losses than for fiber and weather can affect the signal, therefore, many microwave links need a fiber backup.

B. Medium Acquisition Delays

Links can use Medium Access Control (MAC) to control access to a shared medium, which can incur additional link latency. While most wired links use a dedicated point-to-point interconnect, for wireless media, use of a shared radio resource is common (e.g., wireless, cellular mobile, satellite). MAC techniques are also used in shared access networks and for some LAN technologies to share the wired capacity, e.g., data over cable (DOCSIS) and passive optical networks (PON). The design of the MAC can impact the link latency, for example [112] shows that in GPON, the dynamic bandwidth allocation (DBA) mechanism can incur a delay of up to 30 ms when not efficiently done.

There are several fundamental transmission techniques for channel access, i.e., space division multiple access (SDMA), time division multiple access (TDMA), frequency division multiple access (FDMA) and code division multiple access (CDMA)/spread spectrum multiple access (SSMA). Each technique has merits for a specific scenario and type of media. In general, MAC mechanisms may be classified according to the scheduler and multiple-access channel schemes. Transmission latency depends more on the MAC mechanism, than on the channel carrier technique. Reference [113] divides the techniques in three classes: fixed assignment, demand assignment and random access.

Fixed assignment (TDMA and FDMA) provides access to predefined slots (time or frequency), yielding a predictable access latency. However, compared with the other assignment classes, TDMA sources can incur higher latency waiting for their assigned time slot and senders are unable to make use of any unallocated time slots. FDMA sources are restricted to their assigned portion of the frequency spectrum so, similarly, senders are unable to use capacity that has not been assigned.

Demand assignment protocols divide capacity based on requests from devices that generate allocations. A MAC protocol may be centralized or distributed, using polling or reservation for requests. The latency to access the channel comprises the request, allocation and transmission opportunities; the resulting delay depends on: the choice of technique, system configuration and level of traffic. In some systems, requests are not reliably sent, adding further delay if these are lost. Delay may also be incurred if a resource manager determines that a request may be denied or only partially honored, postponing transmission to a later allocation cycle. Cellular mobile [114] and broadband satellite are examples of systems based on such protocols. Sooriyabandara and Fairhurst studied the delay impact on TCP in satellite systems [115].

Random access or contention-based systems operate without allocating dedicated slots/frequencies to a specific terminal. Using random access, transmissions are not coordinated and mostly use fully-distributed access policies, although transmission opportunities may be synchronized to centralized timing to improve stability and utilization. This can provide low latency access to a lightly loaded channel, but latency can increase if there are collisions and a loss recovery mechanism is invoked (e.g., back-off or redundant transmission). In adaptive channel sensing random access, the network device first senses the channel to schedule its message for transmission and better avoid collisions; this is achieved without access coordination with other devices. Wireless systems typically are designed to be adaptive, but the longer propagation delay in satellite broadband often requires other contention resolution methods.

The IEEE 802.11 MAC protocol [116] has become the *de facto* layer-2 standard for wireless local area networks (WLAN). From 802.11e and 802.11n frame aggregation was introduced, defining two techniques for a device to aggregate several MAC frames into a single larger frame, which can reduce delay by reducing the number of contention based media accesses required to send the data. Aggregate MAC Service Data Unit (A-MSDU) allows multiple MSDUs of the same traffic class to be concatenated in a single MPDU destined for a receiver where the sub-frame header corresponds to the same MAC header parameters (RA and TA). In contrast, an Aggregate MAC Protocol Data Unit (A-MPDU) aggregates multiple MPDUs in a single PHY (physical) protocol data unit (PPDU) frame, consisting of sub-frames, each with their own MAC header. In addition, it requires the use of block acknowledgement or Block Ack (BA), whereas instead of transmitting individual ACKs for every MPDU, a single BA frame containing a bitmap field maps and accounts for several MPDUs to be acknowledged.

Frame aggregation improves efficiency by reducing the overhead from MAC headers. Skordoulis *et al.* [117] and Lin and Wong [118] show that dramatic throughput efficiency gains can be achieved using A-MSDU and A-MPDU, whereas there is an optimal aggregate frame size in respect of the noise level of the channel. There is a tradeoff between processing time required to compute the aggregates and increase in overall delays, yet Shen *et al.* [119] show that adaptive frame aggregation can deliver the performance required for a service with a stringent delay requirement, such as VoIP and interactive gaming.

Wireless rate adaptation mechanisms seek to determine the optimal send rate for the current conditions of a varying wireless channel. How well they perform can significantly impact the end-to-end delay of a wireless segment. Early schemes were based on frame loss or Signal-to-Noise Ratio (SNR), with newer techniques combining these signals and MAC control mechanisms to determine the rate of successful frame transmissions [120].

In summary, MAC protocols can introduce significant complexity to share access to the medium. These techniques are often optimized for channel efficiency or throughput, but can also be appropriately optimized to avoid unnecessary latency, and can offer prioritized access to latency-sensitive traffic. However, achieving both at the same time or without the complexities of traffic classification is still an open research issue.

C. Serialization Delay

Serialization/deserialization is the process of getting data from the network card buffer to the wire or vice-versa. At each end of a link serialization delay $S_S = \frac{\text{frame-size}}{\text{line-rate}}$ can be introduced at the sender, and deserializing delay $S_D = \frac{\text{frame-size}}{\text{line-rate}}$ at the receiver. This delay can vary with the path conditions (e.g., adaptation of the rate of a wireless link according to the channel condition). Poor rate adaptation can significantly increase latency [121], [122].

Along the forwarding path each intermediate network device i introduces a deserialization and serialization delay $S_{D(i)} + S_{S(i)}$ every time a frame is read from the wire into memory then written from memory to the wire (with routing/switching in between). Serialization/deserialization introduces delays proportional to the speed of the interface. The delay can be reduced either by increasing the physical layer speed or reducing the number of links that a packet needs to traverse, as shown in virtualizing chains of network functions (see Section II-D4).

Other forwarding methods such as cut-through and wormhole switching [123] reduce serialization/deserialization. Cut-through [124] avoids waiting for a whole packet to be transmit buffered, as in store-and-forward switching; instead, a packet is forwarded as soon as the destination address is identified, thus at each endpoint serialization/deserialization delays remain the same, $S_{S(1)} = \frac{\text{frame-size}}{\text{sender-line-rate}}$ and $S_{D(N)} = \frac{\text{frame-size}}{\text{receiver-line-rate}}$. But at each interior hop, cut-through can reduce the path routing/switching serialization delay to $S_{S(i)} = \frac{\text{header-size}}{\text{sender-line-rate}}$ and deserialization $S_{D(i)} = \frac{\text{header-size}}{\text{receiver-line-rate}}$, thus depending only on the time to serialize the header information. However, in cut-through switching frame errors cannot be detected until the end of the packet, thus corrupted packets are forwarded and could impact network performance.

In wormhole switching, a packet is divided into smaller pieces called flow control digits (or flits). A flit is usually sent from i to $i+1$ only if the device $i+1$ has sufficient buffer space

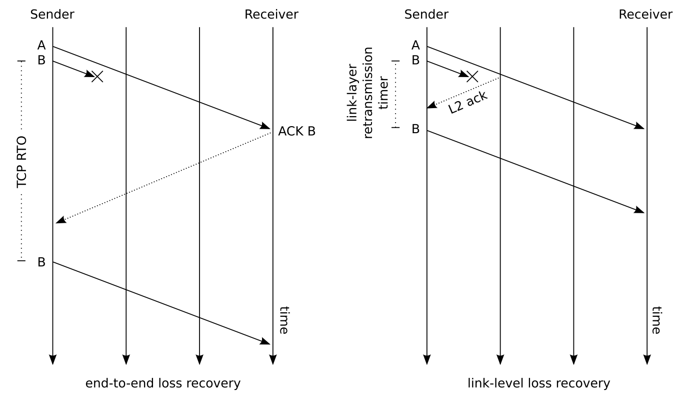


Fig. 14. End-to-end (i.e., TCP-based) versus link-level retransmission. Lack of a link-layer ACK on the first hop triggers local retransmission of the frame carrying TCP segment B.

for the flit. Similar to cut-through switching, once the header flits are received forwarding can be set up, however if the output channel is busy, flow control blocks trailing flits—temporarily buffering them back along the established path toward the sender.

Wormhole switching requires fewer I/O buffers, it enables more efficient port utilization with the implementation of virtual channels, thus reducing latency compared to store-and-forward switching. If intermediate network devices do not discard packets, more reliable transmission is achieved (packets are only lost due to bit errors). On the other hand this introduces more complexity as it may cause deadlocks. Wormhole switching is mainly used in multiprocessor systems in Network-On-Chip systems (NOCs [125], see Section VI-C).

D. Link Error Recovery Delays

Link error recovery delays depend on the compromise made between the link capacity, physical layer coding to avoid errors and link retransmission to correct errors. Shannon [126] showed that for a given available signal bandwidth, the capacity of a link is limited by the channel quality. In simple terms, conservative channel coding can reduce the probability of errors, but long codes can also increase packet serialization delay.

Reliable communication (e.g., using TCP) over a path containing an error-prone link (e.g., a radio-link) can result in occasional appreciable corruption/loss of packets. Link retransmission could detect and recover from channel corruption, but incurs delay (i.e., at least 1 additional *link* round trip of delay—at least 0.5 to notify the packet/frame loss and 0.5 to resend the packet/frame to the link receiver). The loss could also be recovered by end-to-end retransmission by a reliable transport, but this would incur an additional delay of at least 1 times the *end-to-end* path RTT. In practice, the link round trip delay is often much less than the path RTT, and often also supports selective retransmission requests enabling faster loss recovery (Fig. 14). Link retransmission complements transport recovery (especially at times of high loss), but requires careful design to avoid retransmission at multiple levels [92]. It can also introduce unpredictable variation in the path latency.

Cross-layer approaches can help reduce delays caused by the duplication of retransmission at the transport and link layers [127], [128]. Therefore, link design needs to consider the expectations of the types of traffic a link supports, and how that can be optimized for overall latency.

Link FEC is widely used in both wired (e.g., DSL) and wireless technologies to recover from physical layer errors. FEC may be implemented using Reed-Solomon (RS) coding, although higher-efficiency codes (e.g., Low-Density Parity Codes, LDPC) are often used on bandwidth-limited radio links. Coding can be combined with Interleaving (I-FEC) to provide better resilience to bursts of errors (i.e., impulse noise protection (INP) for interference).

There is an intrinsic trade off when choosing interleaving parameters: a more robust scheme with higher delay, or one with less protection but less latency. In DSL, as in radio links, this has a direct impact on the delay of the transmission layer. Interleaving in DSL provides INP protection using I-FEC. A typical INP overhead protection setting introduces 8 ms delay in the downstream direction and a 0 ms delay, with no protection, in the upstream direction [129] over the physical line (see Section IX-D3).

Another way to reduce the need for end-to-end recovery (and hence latency) is to improve the link channel quality (i.e., the signal to noise ratio). This reduces the need for link ARQ or the delay associated with FEC and reduces the latency impacts of wireless rate adaptation (see Sections IV-B & IV-C). This could involve replacing a microwave link with an optical fiber link, or changing the coding and modulation schemes on a radio link.

E. Switching/Forwarding Delay

The links along a network path are connected by network devices, such as core switches and routers. Transit through each device adds to the path latency. There are a number of different network device architectures: input buffered, output buffered, combined input/output buffered, etc., each with different latency characteristics [130]. In a typical design, the forwarding engine examines each packet header to select the appropriate output link. The latency incurred, S_L, is not a constant, but depends on the complexity and the number of rules processed and the speed at which entries in the forwarding table may be accessed. It is often possible to provide a minimum latency, and usually also a maximum latency [131].

Once the output link has been selected, the packet passes through the switching fabric with latency S_F. In a device with several cards (or several racks), S_F will also depend of the relative location of the input and output interfaces. The total latency through the fabric may be:

$$S_{\text{base}} = S_L + S_F \tag{1}$$

In the classical store-and-forward architecture a packet can be queued at the input waiting for the forwarding engine, introducing a latency of S_I, and/or at the output when there is contention, incurring S_O of additional delay (Section IV-F). In addition to this queuing delay in the intermediate network devices, deserialization S_D and serialization S_S introduces further delay that can be addressed using different mechanisms (see Section IV-C). Hence the total switching latency will be:

$$S_{\text{total}} = S_D + S_I + S_L + S_F + S_O + S_S \qquad (2)$$

Software Defined Networking (SDN) allows an operator to implement a set of rules, and can be expected to increase the load on the rule engine, increasing S_L [132]. SDN controller performance is an area of active research [133]. S_{base} can be reduced by increasing the switching fabric speed, lowering S_F.

F. Queuing Delay

Delays from packet queuing at devices along the end-to-end path, in general, contribute the largest delays to the flight latency. This latency originates from contention for either the switching fabric or the output interface (S_I and S_O in Section IV-E). Provisioning sufficient resources will always reduce (or eliminate) contention, but at the expense of decreased link utilization. Input and/or output buffering is needed to ensure high utilization with bursty network traffic, but inevitably leads to building queues. The overall effect of queuing delay is complex, with buffering often present in each device and each network layer or sub-layer. In this section the term *buffer* will refer to the resources available to queue packets, and the term *queue* will refer to the amount of buffer space being used.

Managing queues for quality of service metrics, including latency, was a very active area of research until Dense Wave Division Multiplexing (DWDM) made core network over-provisioning a cost-effective approach to support latency sensitive traffic [134]. More recently, latency and network buffering issues have again received attention through the efforts of Gettys [135], who coined the term *bufferbloat* to describe the over-abundance of buffering resources in several parts of typical Internet paths. Large queues can induce high latency at any congested point on the end-to-end path. Currently this is mainly an issue at the edge of the network [136], [137], but the problem will increasingly affect the core as network access speeds increase.

Efforts to reduce queuing delays along the path can be divided into seven approaches: 1) Flow and circuit scheduling, 2) Reducing MAC buffering, 3) Smaller network buffers, 4) Packet scheduling, 5) Traffic shaping and policing, 6) Queue management, and 7) Transport-based queue control.

1) Flow and Circuit Scheduling: A network device can avoid queuing delays by directly connecting its inputs and output ports (as in, e.g., optical switches used to handle the high rates in the core of the Internet or in data centers [138]). There are many types of optical switching [139], but two main categories: Circuit-switched (wavelength, fiber or time slot) and connectionless (packet and burst). The former requires the *a priori* set up of an all optical path from ingress to egress, resulting in less statistical multiplexing [140]. However, after a path has been established, there is no $S_{[IOL]}$ delay and S_F has also potentially been reduced (see Section IV-E). For data travelling along this path, delay will be the speed of light in the fiber times the distance. If such a path is not available, data may have to wait for a path to be created, or may have to be routed via another egress, resulting in a temporary increase in latency and jitter.

Since only small optical buffers are currently feasible, designs for optical burst and packet switches have almost no buffering delay ($S_{[IO]}$). Currently, optical burst switching is the most practical of the connectionless optical switching techniques [141], [142]. In burst switching, packets destined for the same egress are collected in a burst buffer at the ingress and sent in a group. This reduces or removes the need for buffering in the network, but can increase the overall end-to-end latency due to the additional ingress buffering. Optical packet switches are still an area of active research, and developments may help improve latency compared to burst switching because they do not require the extra ingress buffering of optical burst switching.

Both burst and packet switching may involve tuning wavelength converters, configuring Micro-Electro-Mechanical-System (MEMS) switches, and/or wavelength selective switches. Depending on the architecture, switching delays of the order of 100–300 ns may be achievable resulting in these being no slower than electrical switches [138].

2) Reducing MAC Buffering: Buffering at or below the MAC layer is present for a range of reasons, including: traffic differentiation; header compression; capacity request/ medium access (Section IV-B); FEC encoding/interleaving (Section III-C); transmission burst formation; handover and ARQ (Section III-C). While systems are typically designed for common use-cases, a large number of independently maintained buffers can add significant amounts of latency in ways that may not be immediately obvious [135], [143]: e.g., when traffic patterns change, the radio resource becomes congested, or components of the system are upgraded revealing buffering in a different part of the network stack.

Network devices have moved from a position where under-buffering was common to where MAC buffer bloat can now significantly increase latency [144], with latencies of many seconds not uncommon in an un-optimized system.

Delays can also result as a side effect of other link protocols. For example, some Ethernet switches implement a type of back pressure flow control using IEEE 802.3X PAUSE frames [145]. If the input queue of a downstream switch is full, a switch can send a PAUSE frame to cause upstream devices to cease transmission for a specified time. This mechanism can avoid loss during transient congestion, but sustained overload can result in a cascading effect that causes the buffers of switches along a path to be filled—dramatically increasing the path latency [146]. Anghel *et al.* [147] show that use of PAUSE frames in data centers can improve flow completion times, but that care is needed in setting the thresholds in switches and ensuring that there is end-system support. Priority Flow Control (PFC) is an enhancement that can reduce delay for latency sensitive flows by allowing the PAUSE to specify a particular class of traffic in IEEE 802.1Qbb [148].

In general, unnecessary buffering below the IP level needs to be eliminated to improve latency. Where possible, packets should be buffered in the IP layer using Active Queue Management (AQM) methods [149] (Section IV-F6). This can require a redesign of the architecture to enable coordination between protocol entities, avoiding the pitfalls of direct implementation of a large number of independent layers and construction of individually buffered "pipes/streams" across the lower layers [135], [143], [150], [151]. In the Linux kernel since Aug. 2011, Byte Queue Limits (BQL) has significantly reduced delay within network interface hardware, but without radically altering inter-layer co-ordination [152]. It is hard to size a hardware transmit buffer statically, because it consists of a queue of pointers to packet buffers that may each range in size from a few bytes to 64 KiB. So BQL does not attempt to reduce this hardware buffer size. Instead it separately maintains a record of bytes queued and it dynamically adapts a byte limit to track the variation in queue size using a new Dynamic Queue Limits (DQL) library [153]. Whenever the bytes queued exceeds the limit, further queuing from the upper layers is blocked. This minimizes the lower layer queue, and pushes any larger queue up into the network stack where AQM can be applied.

3) Smaller Network Buffers: The most effective means of reducing queuing delay is to reduce the size of buffers in each device along the end-to-end path, this limits the maximum queue size. An early buffer dimensioning rule-of-thumb [154] recommended that buffers should be sized proportional to the line rate times the RTT ($B = \text{RTT} \times C$), the Bandwidth Delay Product (BDP), but this is now known to be excessive.

Appenzeller *et al.* [155] investigated whether BDP sized buffers are required for high utilization in core routers, and showed that core router buffers can take advantage of a high degree of statistical multiplexing and reduce BDP sized buffers by a factor of $\sqrt{n}$, $B = \frac{\text{RTT} \times C}{\sqrt{n}}$, where n is the number of concurrent flows on the link. Further reductions are possible if the full utilization constraint is relaxed, though Dhamdhere and Dovrolis [156] raised concerns of higher loss rates and thus lower TCP throughput. The work by Appenzeller *et al.* and an update [157] spawned a number of studies and proposals. Vishwanath *et al.* [158] surveyed and critiqued much of this work and conducted experiments with mixed TCP and UDP traffic. They concluded that small buffers make all-optical-routers more feasible. As well as reducing latency, Havary-Nassab *et al.* [159] showed that small buffers make the network more robust against Denial-of-Service (DoS) attacks.

Although work on reducing buffer sizes in the core network is necessary and important for the future, most current congestion is closer to the network edges, where there is not a high degree of statistical multiplexing. Chandra [160] divides congestion into packet-level and burst-level congestion. Packet-level congestion only requires small buffers, however congestion due to traffic burstiness and correlations requires much larger buffers. For this reason, small buffers at lightly multiplexed network edges require traffic to be smoothed or paced to avoid burst-level congestion and allow smaller buffers (Sections IV-F5 and IV-F7).

Optimizing buffer sizes for various scenarios is still an area of research. A trade-off will remain between latency, utilization and packet loss—with latency expected to become more critical.

4) Packet Scheduling: Packet scheduling can also impact latency. A scheduling mechanism allows a network device or endpoint to decide which buffered packet is sent when multiple packets are queued. Internet hosts and network devices have by default used first-in-first-out (FIFO) scheduling, which sends the oldest queued packet first. This can cause head-of-line blocking when flows share a transmission link, resulting in all flows sharing an increased latency. There are, however, a wide variety of queue scheduling mechanisms and hybrid combinations of mechanisms that can either seek to ensure a fair distribution of capacity between traffic belonging to a traffic class/flow (class/flow isolation), or to prioritize traffic in one class before another. These methods can reduce latency for latency-sensitive flows. This section does not seek to explore all scheduling methods, but will highlight some key proposals.

a) Class based: Some scheduling mechanisms rely on classifying traffic into one of a set of traffic classes each associated with a "treatment aggregate" [161]. Packets requiring the same treatment can be placed in a common queue (or at least be assigned the same priority). A policy apportions the buffer space between different treatment aggregates and a policy determines the scheduling of queued packets. Different classes of traffic may receive a different quality reflecting their latency and other requirements, so scheduling with this knowledge can have positive impacts on reducing latency for latency-sensitive flows.

A router or host-based model implements scheduling in individual routers/hosts without reference to other devices along the network path. This is easy to deploy at any device expected to be a potential bottleneck (e.g., a home router), although it does not itself provide any end-to-end quality of service (QoS).

A more sophisticated model aligns the policies and classes used by the routers across a domain, resulting in two basic network QoS models: The differentiated services model [162] aligns the policies configured in routers using the management plane to ensure consistent treatment of packets marked with a specific Differentiated Services Code Point (DSCP, [163]) in the IP packet header (i.e., devices schedule based only on treatment aggregates). In contrast, the integrated services model [164] uses a protocol to signal the resource requirements for each identified flow along the network path, allowing policies to be set up and modified in real-time to reflect the needs of each flow. Both models can provide the information needed to control the delay of traffic, providing that delay-sensitive traffic can be classified. The integrated services model is best suited to controlled environments, e.g., to control latency across an enterprise domain to support telepresence or other latency-sensitive applications.

b) Flow based: Flow based queuing allows a scheduler to lessen the impact that flows have on each other and to discriminate based on flow characteristics. A key example of this is Fair Queuing (FQ), proposed by Nagle [165] and Demers *et al.* [166], which aims to improve fairness among competing flows. This ensures that queuing delays introduced by one (possibly misbehaving) flow do not adversely affect other flows, achieving fairness. FQ has been adapted and extended in many different ways including: weighted FQ [166], [167], and practical approximations such as Round Robin (RR) scheduling and Deficit RR (DRR). In practical implementations, there may be a limit to the number of queues that can be implemented, hence stochastic fair queuing (SFQ [168]) and

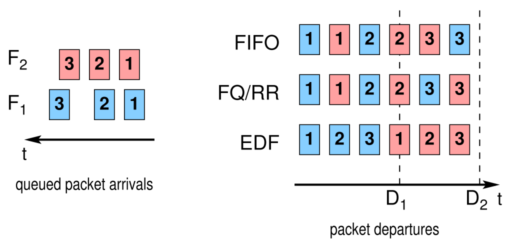


Fig. 15. Sharing of available capacity by two flows, illustrating the difference between FIFO, FQ/RR and EDF scheduling. Flow F_1 has deadline D_1, and flow F_2 had deadline D_2.

similar methods have been proposed to eliminate the need for a separate queue for each traffic flow.

Per-flow classification requires a flow classifier to discriminate packets based on the flows to which they belong and deduce their required treatment. In the router or host-based model this function is performed at each router and requires visibility of transport protocol headers (e.g., protocol and port numbers), whereas in the Differentiated or Integrated models visibility is required at the edge of the QoS domain. When tunnels are used, all tunnel traffic is generally classified as a single flow (an exception could be the use of the IPv6 Flow Label to identify subflows). This can add latency by assigning all traffic that uses a VPN tunnel to the same queue, besides the obvious processing cost of encryption and decryption.

c) Latency specific: Some schedulers schedule packets to achieve a low or defined latency. The simplest is Last-In-First-Out (LIFO [169], [170]), which minimizes the delay of most packets—new packets are sent with a minimum delay at the expense of packets already queued. Unfortunately, this method also maximizes the delay variance and reorders packets within a flow.

Deadline-based schemes attempt to bound the latency of a queue, e.g., Earliest Deadline First (EDF [171]), where jobs (or packets [172]) are scheduled in the order of their deadline. The principle of EDF scheduling is illustrated in Fig. 15, for two flows with different deadlines; both flows can meet their deadlines if the flow with the earliest deadline is scheduled first. Unfortunately, these methods fail to provide good performance under overload.

Shortest Queue First (SQF) is a flow/class based scheduler that serves packets from the flow/class with the smallest queue first. It has been proposed for reducing latency in home access gateways [173]. Carofiglio and Muscariello [174] show that the SQF discipline has desirable properties for flows that send less than their fair share, such as thin latency-sensitive flows and short flows, at the expense of bulk throughput-sensitive flows.

d) Hierarchical scheduling: In many networks it is normal to create a hierarchy of scheduler treatments, in which some classes of traffic are given preferential or worse treatment by the scheduler, to achieve different treatments for the traffic. For example the Expedited Forwarding (RFC 3246) differentiated services class assigns a treatment that offers low loss and low latency. Class-based queuing [175], hierarchical packet fair queuing [176], hierarchical fair service curve [177] and 802.11e QoS enhancements for WLANs [178] provide such methods. Any priority-based algorithm needs to correctly classify the traffic in a way that guarantees the required treatment. This typically requires a policing (or traffic-conditioning) function to prevent misuse. In the integrated and differentiated services model this conditioning may be provided at the domain edge [179].

5) Traffic Shaping and Policing: Traffic shapers smooth traffic passing through them using a buffer to limit peak transmission rates and the length of time these peak rates can be maintained. While shaping can help prevent congestion—and therefore delay further along the path—, it does so at the expense of delay introduced at the shaper. The foundational traffic shaping algorithms are the leaky bucket algorithm (Turner [180]) and the related token bucket algorithm. Traffic shapers are used extensively in the Internet [181], though often to reduce ISP costs rather than to reduce delays along the path [182].

Traffic policers drop packets that exceed a specified peak transmission rate, peak burst transmission time, and/or average transmission rate. Policing was first proposed by Guillemin *et al.* [183] for ATM networks, but is still an effective tool for managing QoS, especially latency, in the Internet. Briscoe *et al.* [184] propose a policing mechanism that could be used either to police the sending rate of individual sources (e.g., TCP) or, more significantly, to ensure that all the sources behind the traffic entering a network react sufficiently to congestion, as a combined effect, without constraining any flow individually. This developed into the IETF Congestion Exposure (ConEx [185]) work to enable a number of mechanisms, especially congestion-based policing, to discourage and remove heavy sources of congestion and the latency they cause.

6) Queue Management: Network devices can monitor the size of queues and take appropriate action as the queue latency builds; this is known as queue management. Techniques such as *drop tail* and *drop front* [186] are said to be *passive*. In contrast, Active Queue Management (AQM) techniques manage queues to achieve certain queue loss and latency characteristics by proactively marking or dropping packets; which signal to endpoints to change their transmission rate. AQM mechanisms generally work in combination with scheduling, traffic shaping, and transport-layer congestion control. Adams [187] provides an extensive survey of techniques from Random Early Detection (RED [188]), introduced in 1993, through to the year 2011. This section looks at more recent contributions, with a specific focus on latency.

a) PIE and CoDel: Two current proposals aim to minimize the average queuing delay: Proportional Integral controller Enhanced (PIE [189]) and Controlled Delay (CoDel [190]), and more recently, a per-flow queuing version of CoDel called FQ-CoDel.

The PIE algorithm uses a classic Proportional Integral controller to manage a queue so that the average queuing delay is kept close to a configurable *target delay*, with a current default value of 20 ms. PIE does this by using an estimate of the current queuing delay to adjust the random ingress packet drop or marking probability. The algorithm self-tunes its parameters to adapt quickly to changes in traffic. PIE tolerates bursts of

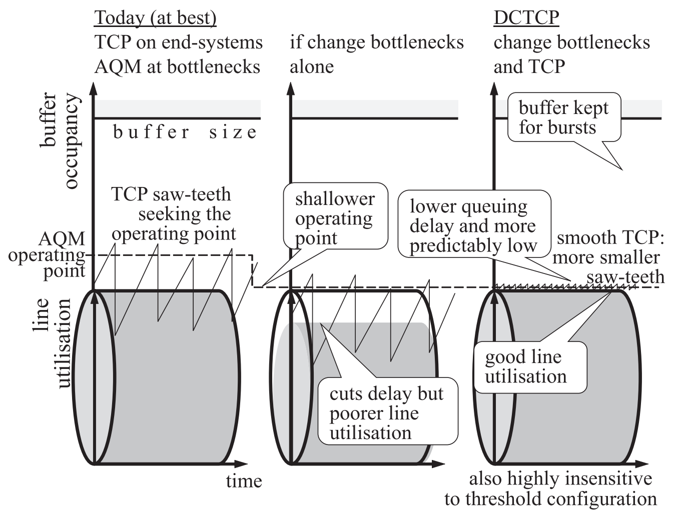

Today (at best)
TCP on end-systems
AQM at bottlenecks
if change bottlenecks alone
DCTCP
change bottlenecks and TCP
buffer occupancy
buffer size
buffer kept for bursts
TCP saw-teeth seeking the operating point
AQM operating point
shallower operating point
lower queuing delay and more predictably low
smooth TCP: more smaller saw-teeth
line utilisation
good line utilisation
cuts delay but poorer line utilisation
time
also highly insensitive to threshold configuration

Fig. 16. How Data Centre TCP (DCTCP) reduces delay without losing utilization.

packets up to a configurable maximum, with a current default value of 100 ms.

CoDel attempts to distinguish between two types of queues, which the authors refer to as *good queues* and *bad queues*—that is, queues that simply buffer bursty arrivals, and those just creating excess delay. Although the default target delay is 5 ms, it allows temporary buffering of bursts which can induce delays orders of magnitude larger than the target delay. Packets are dropped or marked at deterministic intervals at the head of a queue. Dropping/marking at the queue head decreases the time for the transport protocol to detect congestion. Combining this with the flow isolation of a fair queuing scheduler (see Section IV-F4) avoids packet drops for lower-rate flows.

Both schemes attempt to keep configuration parameters to a minimum, auto tune, and control average queue latency to approach a target value. Both exhibit high latency during transient congestion episodes. Use of smaller buffers would prevent this, but this is an area that requires further research (see [191]–[193]).

b) DCTCP and HULL: Data Centre TCP (DCTCP [194]) is illustrated in Fig. 16. It uses an AQM method that has been designed to keep queuing delay and delay variance very low, even for small numbers of flows including a single flow. The method appears deceptively simple; it merely marks the ECN field [63] of all packets whenever the queue exceeds a short threshold.

The AQM for DCTCP signals even brief excursions of the queue, in contrast to other AQMs that hold back from signaling until the queue has persisted for some time (see Section IV-F7c). Even though existing switches often only implement RED, they can avoid introducing signaling delay by simply setting their smoothing parameter to zero. The transport-layer functions of DCTCP are described in Section IV-F7.

High bandwidth Ultra-Low Latency (HULL [195]) replaces the AQM algorithm in DCTCP. It aims to keep the real queue extremely short by signaling ECN when a *virtual queue* exceeds a threshold. A virtual queue is a token-bucket-like counter that fills at the real packet arrival rate, but drains slightly more slowly than the real line. A growing range of commercial equipment natively supports virtual queues, often using two hardware leaky buckets [196].

7) Transport-Based Queue Control: A number of transport layer mechanisms have been proposed to support low queuing delays along the end-to-end path. Two key elements are: burstiness reduction and early detection of congestion.

a) Coupled congestion control: When multiple flows that originate from the same endpoint traverse a common bottleneck, they compete for network capacity, causing more queue growth than a single flow would. Detecting which flows share a common bottleneck and coupling their congestion control (as, e.g., in [197]) can significantly reduce latency, as shown with an SCTP-based prototype in [198]. Solutions in this space are planned outcomes of the IETF RMCAT working group [199] (see also Section V-C2).

b) Burstiness reduction: A TCP session that always has data to send typically results in paced transmission, since the sender effectively paces the rate at which new data segments may be sent at, to the rate at which it receives ACK packets for old segments that have left the network. However, this is not always the case. TCP's window-based congestion control together with bottleneck queuing can result in very bursty traffic [200]. In some implementations, the TCP max_burst function [201] limits the maximum burst size per received ACK, hence reducing burstiness for bulk applications.

However, not all applications continuously have data to transmit (e.g., when a server responds to requests for specific data chunks, or when a variable rate video session experiences a scene change). There may therefore be periods in which no TCP data are sent, and hence no ACKs are received—or an application may use an entirely different transport that does not generate an ACK for every few segments. Either of these can result in bursts of packets, and may require explicit pacing at the sender or a traffic shaper within the network.

Host pacing is a method that limits the burstiness of a flow (similar to host-based traffic shaping). It can result in smaller network queues, reduced buffering requirements (see Section IV-F3), and also reduce the latency experienced by flows that would otherwise be queued behind the bursty traffic [202]. However, the effects of TCP pacing on the network are complicated and depend on many factors (Aggarwal *et al.* [203]). Wischik [204] shows that it has the potential to decrease stability unless buffers are small. Wei *et al.* [205] conclude that the effect of pacing is generally positive, improving the worst-flow latency. Enachescu *et al.* [206] propose a TCP pacing scheme that may allow network buffer sizes to be very small, even on the order of 10–20 packets in core routers.

c) Low threshold ECN: An explicit signal of the network load or congestion, before, or just as, the congested queue begins to grow provides the best signal to a transport layer. Unfortunately, the current standard for ECN [63] does not define such a signal. Instead, ECN signals are treated equivalently to loss signals, both when generated in the network and when responded to by end systems.

A form of *instantaneous* ECN marking (see Section IV-F6) was designed for Data Centre TCP (DCTCP [194]), which smooths the congestion signals at the endpoints:

- End-systems smooth signals over their own RTT, avoiding the network needing to conservatively delay signals for the longest likely RTT, holding back information from short-RTT flows for tens or even hundreds of their RTTs.
- The rate-response of end-systems can be proportionate to the extent of the signals, not just a fixed conservatively large rate reduction (i.e., halving) in response to a signal (see Fig. 16).

In DCTCP, a congestion signal α is computed by the sender, measuring the fraction of ECN-marked packets in a window. DCTCP's congestion control uses α to drive how the congestion window, `cwnd`, evolves. In the absence of congestion (i.e., no ECN marks) `cwnd` grows as in standard TCP, whereas `cwnd` reduction is proportional to the level of congestion α; halving the window only happens under very heavy congestion ($\alpha = 1$). A DCTCP source can therefore maintain a small-amplitude sawtooth that closely hugs the available capacity, allowing the AQM threshold to be very short, therefore keeping delay low and near-constant.

DCTCP was deployed by Microsoft within its Bing data centers, and in Windows 8. The Linux implementation has been modified slightly to work over long RTT paths. Results in [194] show that it behaves well with the shallow buffers present in commodity switches, and yields much fewer timeouts. Achieving both low delay and high throughput is at the expense of slower convergence time for long flows; this is due to the less aggressive congestion response compared to TCP.

DCTCP spawned other recent transport protocol advances [195], [207], [208] that leverage ECN and AQM to reduce queuing delay. All these approaches are currently restricted to environments like data centers and private networks, where a single administrator can ensure all endpoints and network devices universally support ECN and where congestion signals are responded to smoothly in all end-systems, and not delayed in the network. There is ongoing research on how to enable the DCTCP approach to co-exist with existing traffic in public networks [209].

Deadline-Aware Data Center TCP (D^2TCP [207]) relies on instantaneous ECN markings and maintains an estimate of the level of congestion α just as DCTCP. The reaction to impending congestion is modulated according to deadlines: near-deadline flows back off much less (or even not at all) compared to far-deadline flows. This is achieved by a slight change to the DCTCP congestion response, where α is replaced by α^d, with $d > 0$ an indication of how far the flow's remaining time is from its deadline; for long flows without an explicit deadline, $d = 1$ and D^2TCP behaves exactly like DCTCP, whereas $d > 1$ for flows which deadline is near.

L^2DCT (Munir *et al.* [208]) also draws on DCTCP, but differs in the way the congestion window is updated. It is an end system change that emulates a Least Attained Service scheduling discipline, which prioritizes short flows without a need to know their length or deadlines *a priori*; only the amount of data already sent is used. The congestion response follows that of D^2TCP, but with a system of "weights" that depend on the amount of data sent. When there is congestion, L^2DCT makes long flows back off more than short flows. When there is no congestion, long flows increase their congestion window more conservatively than short ones, with the amount of increase progressively getting smaller for longer flows (down to a minimum value).

HULL [195] leverages the ideas behind DCTCP (see Section IV-F6). HULL aims at operating with *almost-empty buffers*. Because it can leverage a virtual queue, HULL trades bandwidth for latency; the loss of bandwidth is tunable by adjusting the parameters of the virtual queue. HULL trades lower latency for short flows against longer completion times for long-lived flows. Since there is no ACK clock, HULL also requires a pacing module at the sender to dissipate burstiness.

These proposals achieve significant improvements in latency performance with respect to standard TCP (even when standard TCP is used in combination with RED and standard ECN). For instance, according to Alizadeh *et al.* [194], DCTCP allows the network to carry ten times as much data (both background and queries) while allowing an $\sim$80% reduction in the 95th percentile of query completion times, with timeouts being close to non-existent (0.3% of flows). With realistic traffic patterns, 99.9th percentiles of completion times for short flows (queries) can be reduced by as much as $\sim$40%. Other proposals claim yet larger improvements. For example, D^2TCP seems to reduce the proportion of flows missing their deadlines by as much as 75%, compared to DCTCP, whereas L^2DCT brings mean completion times down by 50% and 95%, compared to DCTCP and TCP respectively.

d) Delay-based congestion detection: One issue that contributes to queuing latency is that congestion is often not detected until it has already induced delay. Early detection of the onset of congestion allows congestion controllers to reduce their rate quickly before queues build. This could use delay-based inference of congestion or early explicit notification of congestion by the network. Delay based mechanisms can directly measure the quantity that needs to be reduced. They also do not require any specific signaling from the network. Jin *et al.* [210] argue that without an explicit signal from the network, delay is a viable congestion measure. However, despite their long history (their origins stretch back to 1987 when NETwork BLock Transfer (NETBLT [211]) was proposed, with TCP Vegas (Brakmo and Peterson [212]) being the most notable early proposal) delay-based congestion control mechanisms have proven difficult to deploy over the Internet. This is mainly due to potential starvation by conventional loss-based algorithms. To date, Low Extra Delay Background Transport (LEDBAT [213]) is the only widely adopted mechanism (see Section V-D).

Hayes and Ros [214] discuss these co-existence issues arguing that delay-based congestion control should still be considered part of the solution for reducing end-to-end path latency. Delay-based congestion control seems to require no changes to network equipment. However, it only keeps queuing delay low if the traffic can be isolated from competing conventional loss-based algorithms. This could be achieved if the delay-based approach were deployed universally in a closed network, but in the Internet it only works if the current bottleneck queue provides flow separation. Arguments for and against flow separation are outside the scope of the present work, but anyway, a delay-based transport implementation has to assume that not all network queues will protect it from competing flows. Therefore some delay-based approaches accept that they cannot always keep delay low, and they switch to a more aggressive mode if they detect losses (e.g., see [215]).

V. Delays Related to Link Capacities

The available link capacity along a path as well as the way in which the available capacity is used can contribute to the overall latency. Whenever the link capacity is low, packet serialization times may represent a non-negligible component of the total latency of an end-to-end communications chain. Furthermore, capacity is often shared and only a small proportion of the capacity may be available to a specific flow; such is the case when links are congested. Increasing the link capacity would alleviate this congestion and hence reduce queuing delays. When capacity is scarce, delays can also be shortened by reducing the amount of information that needs to be transferred over the network.

Due to protocol inefficiencies, increased capacity may not necessarily yield lower latency. One reason can be TCP congestion control during the initial slow-start phase, which may often not be able to reach the available capacity before the end of a flow, making the duration of the transmission solely dependent on the RTT. While protocols should efficiently use available capacity, they must also be aware of other flows, ensuring that limited capacity is shared in a way that avoids inducing delay or loss to the other flows. Collateral damage caused to other flows can be limited by yielding capacity to higher priority flows or by avoiding to inject large bursts of data into the network.

The details of these issues are further explored in this section.

A. *Insufficient Capacity*

Increasing link capacity (i.e., transmission speeds) seems like an obvious way to reduce packet serialization times and the incidence of persistent congestion. This can be achieved by upgrading the physical link interface (e.g., 10 GE to 100 GE). However, such a brute-force approach to reduce latency is not always economically sensible, nor technically feasible. An alternative way to increase the available capacity is to use several parallel links or multiple paths at once.

1) Leveraging Multiple Links/Interfaces: At the network layer, a router/switch can bundle/aggregate parallel physical links to form one logical link, e.g., using the Link Aggregation Control Protocol (LACP [216]). Another way is to permit routers to utilize parallel paths, such as in ECMP (see Section II-A), in which the packet forwarding is routed and load-balanced over multiple paths. A third possibility is to simultaneously utilize the multiple network interfaces available on many devices (e.g., using one service to minimize delay, another to maximize throughput or minimize cost), as described by the IETF Multiple Interfaces work [217], [218].

Transport protocols can also support multiple paths. A *multipath* endpoint can dynamically exploit multihoming by striping data across several interfaces simultaneously. This approach is followed by protocols like Multipath TCP (MPTCP [219]–[221]) and Concurrent Multipath Transfer for SCTP (CMT-SCTP [222]), as well as related proposals like Dynamic Window Coupling [223] that may apply to both MPTCP and CMT-SCTP. The main goal of these mechanisms is to maximize throughput—and to balance traffic load across paths—by regarding all available links as a single, pooled resource, with lower transfer times as a result. Further, mixing multipath transmission with some form of end-to-end redundancy (for faster loss recovery) can be envisioned as a means of attaining lower latency [224], [225].

Multipath transmission may reduce transfer times, but in some cases this only benefits long-lived flows. For instance, with MPTCP there is a protocol overhead from setting up a subflow across a path other than the primary one [11]—such overhead adds latency that may not make it worthwhile to use more than one interface for a short flow. Large differences in RTTs add to this delay, if, e.g., the first subflow goes along the shortest RTT path, the transfer may finish before the handshake on the longest RTT path has completed [226]. Another latency-related issue with multipath transports is packet reordering at the receiver [227]. Use of several paths may result in packets arriving out of order, and large differences in path RTTs may increase the severity of reordering events.

Adequate scheduling for concurrent multipath transmission may help to avoid reordering, while at the same time minimizing delivery delay. Multipath scheduling has been studied (e.g., [228]–[231]), although these studies tend to focus on bulk data applications, and/or on throughput optimization. Section IV-F4 provides details of how packet scheduling affects latency.

B. *Redundant Information*

Some types of content are simultaneously requested/delivered to many users, for example Live TV video or massive software upgrades. Instead of redundantly transporting the content over the network, multicast can offer a saving of $\sim 1/n$ in capacity (i.e., a unicast flow sent n times only needs to be sent once for each branch of a multicast tree, replicating the flow at each router/switch). In general, IP multicast is the most scalable solution for delivering high rate services (e.g., high definition broadcast TV), but it is often deployed as a local managed service (i.e., not end-to-end through the Internet). Multicast can also be used for other services, e.g., content cache distribution to populate CDN servers [232] (see Section II-C1), especially for delivering live multimedia content over the Internet [233], [234].

Application Layer Multicast (ALM) avoids the need to deploy IP multicast by creating an overlay network. Automated Multicast Tunneling (AMT) is an alternate technology that mimics end-to-end IP multicast by dynamically creating tunnels to native multicast routers.

For some applications, protocol headers may induce a non-negligible overhead, dominating packet transmission times (for low transmission rates). In such cases, *header compression* can reduce the required capacity. This reduces the transmission latency. Header compression may be applied to one or more layers of the protocol stack, and may be either end-to-end, on a given link, or via a network middlebox.

SPDY (Section VII-A) is an example of use of end-to-end header compression for a specific application protocol (HTTP) that can bring about noticeable delay savings.

Two examples of per-link, multi-layer compression are Compressed TCP [235] and IPHC [236]. Robust Header Compression (ROHC) provides a family of methods that can be used to reduce the size of the IP header as well as upper-layer headers, be it, e.g., TCP [237], UDP and RTP [238], and other protocols. These mechanisms can be very efficient: headers can be compressed to as little as ~10% of their original size [237]; for a TCP ACK segment, this means decreasing transmission times by ~90%. A disadvantage of link-layer header compression is that a single packet loss may increase latency, if the loss results in desynchronization between the state (or "context") of the compressor and the decompressor. For instance, with schemes such as Compressed TCP, context desynchronization may typically inhibit fast retransmit, thus resulting in a TCP timeout that would not happen in the absence of compression [237].

C. Under-Utilized Capacity

It is not straightforward to optimize utilization of the network capacity. On the one hand, the link(s) between two network devices can be monitored and upgraded as described in Section V-A, on the other hand optimal utilization of the network capacity depends on the dynamics of end-to-end traffic flows. This section focuses on how mechanisms for protocols that use congestion control can be improved to reduce transfer latency. It describes techniques for recovering more efficiently from a congestion phase, and mechanisms that enable faster sensing of available capacity to more quickly utilize a network path.

1) More Aggressive Congestion Control: It has been known for more than a decade that bulk flows using standard TCP congestion control [239] perform poorly over paths with a large bandwidth–delay product (BDP). One of the main reasons is the additive-increase, multiplicative-decrease (AIMD) behavior of TCP. In congestion avoidance, a TCP sender complying with RFC 5681 [239] increases its congestion window roughly by one full-sized segment per RTT; following congestion (loss), the sender divides its window by two (at least). Hence, when the BDP is large, a standard TCP sender may need many round-trip times after a congestion event to attain a large window again [240]. This behavior results in a sender being unable to efficiently exploit the available capacity and, thus, in long flow completion times.

More aggressive congestion-control algorithms have been proposed, such as HSTCP [240], Scalable TCP [241], and CUBIC (adopted as the default TCP variant in Linux since kernel version 2.6.15) [242]. These algorithms all modify the additive-increase, multiplicative-decrease (AIMD) rules: by increasing the window growth rate and decreasing the amount of window reduction after a loss, they aim at speeding up window recovery after loss events. The explicit focus of all such proposals is not in low latency, but in maximizing throughput for long-lived flows; shorter completion times come as a by-product.

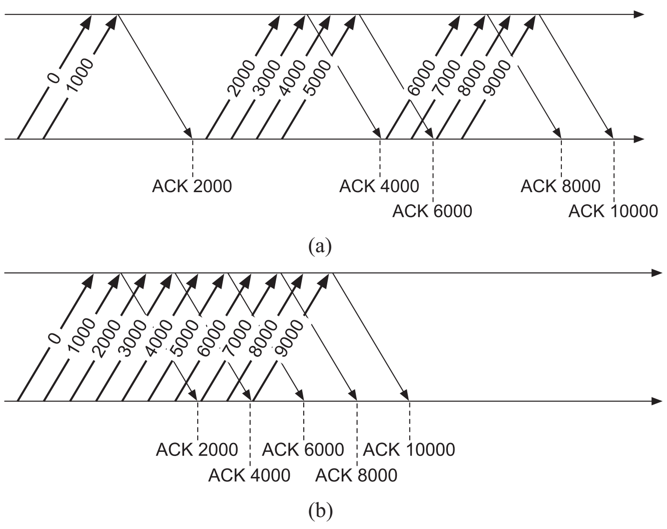


Fig. 17. Impact of the initial window (IW) on transfer times, for two values of IW. (a) IW = 2. (b) IW = 10.

A slightly different approach has been adopted by protocols such as FAST TCP [243] or Compound TCP [244], which use the estimated end-to-end *queuing* delay to control the sender's congestion window. Their goal is to send at the highest possible rate while keeping the amount of queued packets—hence, queuing delay—bounded. (See also Section IV-F7.)

2) Rapidly Sensing Available Capacity: At the start of a connection, or after a long idle period, congestion controlled flows are faced with the challenge of how to efficiently acquire available capacity and determine a suitable safe sending rate that does not adversely impact other flows that share a part of the same path. TCP and SCTP use the Slow Start algorithm [239] to probe for available capacity. Slow Start linearly increases the congestion window for every ACK received, roughly doubling the window each round trip time, leading to an exponential growth of the sending rate. The algorithm continues until congestion is detected or a slow-start threshold is reached. The initial congestion window (IW) standardized in RFC 5681 [239] for TCP and RFC 4960 [61] for SCTP, ranges from 2 to 4 segments depending on the maximum segment size. The small initial congestion window requires several RTTs to transfer even relatively short flows, contributing to latency. The exponential growth of the sending rate in the slow start phase may also lead to a severe overshoot of the available capacity and excessive packet loss [245]. Decreasing the time for TCP to send the initial part of a transfer can significantly reduce latency for applications that send small to medium volumes of data.

a) Sensing capacity without network assistance: In [246], Dukkipati *et al.* studied the effects of using an increased initial congestion window for Web download against geographically spread data centers. Their study suggested that an increased initial congestion window could decrease latencies for this type of transaction. The IETF recently issued an experimental RFC that allows an initial congestion window of up to 10 segments (IW10 [247]). The effect of using higher values of IW on the transfer times of short flows is illustrated in Fig. 17, for a flow of size 10 segments and for two values of IW (2 and 10, respectively). Host pacing may be used in combination with IW10 to reduce the probability of inducing packet loss [248].

The effects of a much more significant change to the TCP start-up phase were earlier explored by Liu *et al.* in a proposal called Jump Start [249]. Jump Start injects all data available in the send queue, up to the size of the receiver's advertised window, already in the first transmission round which in effect allows an unlimited initial congestion window. This data is paced by the end-host using an RTT estimate from the three-way handshake. If data is lost, then Jump Start falls back on the traditional TCP recovery and reduces the congestion window appropriately. How the network would cope with the excessive initial send rate that may result from Jump Start is unclear from the paper, and Jump Start is also shown to be too aggressive in [250].

Several techniques have proposed ways to estimate the sending rate that may safely be sustained by a path. Swift Start [251] uses the packet-pair technique [252] for an initial estimate of the available capacity. To guard against imprecise measurements, a fraction of the estimated capacity is then used as the send rate for the next round. The capacity estimate is combined with pacing to avoid sending a large burst of data into the network. RAPID [253] uses a more advanced scheme where the sending times of the packets are carefully scheduled to probe for multiple possible sending rates in a single RTT. This is achieved by sending the packets within a multi-rate probe stream (or chirp) where the rate increases with time. The available capacity is then estimated based on observations of the inter-packet spacing at the receiver. Although RAPID mainly targets the congestion avoidance phase, it also includes an improved slow-start. Under ideal conditions RAPID allows a suitable initial sending rate to be detected in only 1–4 RTTs. Issues and possible solutions for how congestion control based on chirping can be deployed in production systems are discussed in [254].

An alternative strategy to gaining knowledge about the path is through sharing of information between connections to the same destination, or more generally between connections known to pass through a shared bottleneck. RFC 2140 [255] discusses both the possibility for temporal sharing, where a new TCP connection could reuse the congestion window of an earlier, now closed TCP connection, and the possibility of ensemble sharing, where the congestion state is shared between several active TCP connections. With ensemble sharing a new TCP connection could immediately be given a share of the available capacity towards a given destination. The congestion manager framework described in RFC 3124 [256] generalizes the concept of ensemble sharing to allow joint congestion management across all transport protocols and applications. Current standardization work within the IETF RMCAT working group [257] on congestion control for interactive real-time media also includes the possibility of joint congestion management, with [197] as a first proposal. Although joint congestion management can allow a new flow to quickly find a suitable sending rate, it is only applicable when multiple flows share a common bottleneck, something which is also hard to detect reliably (see also Section IV-F7a).

b) Sensing capacity with network assistance: While the proposals described above did not rely on any network assistance, there are also a number of proposals that rely on advanced network functionality or explicit feedback from the network. TCP Fast Start [258] uses a possibly stale congestion window from previous connections during start-up. To compensate, TCP Fast Start sends packets with a class of service that it expects the network to discard preferentially, in order to protect other traffic. TCP-Peach [259] uses probe packets that are marked to be treated by the network with lower priority in order to detect spare capacity in a satellite network context. Recursively cautious congestion control (RC3) [260] also relies on priority queuing being available in the network. In addition to sending high priority packets following TCP's normal algorithm, it sends additional packets (starting from the end of the flow) at several lower priority levels to take advantage of spare capacity.

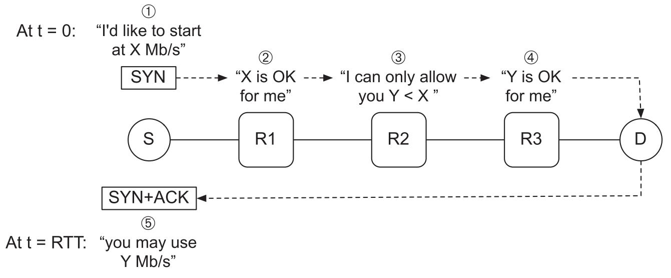

①
At t = 0: "I'd like to start at X Mb/s"
SYN
② "X is OK for me"
③ "I can only allow you Y < X"
④ "Y is OK for me"
S
R1
R2
R3
D
SYN+ACK
⑤
At t = RTT: "you may use Y Mb/s"

Fig. 18. Quick-Start operation. The requested sending rate (X) is acceptable to router R1, but not to router R2 who suggests a lower value (Y), and this value is acceptable to router R3.

The Quick-Start extension to TCP and DCCP [261]–[263] is an example of an approach that relies on explicit feedback from the network. This allows an endpoint to signal a desired sending rate in the TCP SYN segment (Fig. 18). Each router along the path must then agree to the desired rate or reduce the rate to an acceptable value. The receiver feeds the information back to the sender which sets the initial congestion window according to the result.

The eXplicit Control Protocol (XCP [264]) proposes a similar idea, but here each data packet carries a new congestion header. A sender indicates its desired congestion window for the next round in the header and the routers accept or reduce the window size. As XCP receives continuous per-packet feedback on the path congestion, it can not only use available capacity quickly at flow start-up, but also react quickly to changes in traffic load. This maintains a small standing queue size and reduces packet drops.

The Rate Control Protocol (RCP [265]) also proposes a new header field with information allowing routers to add information about the allowed sending rate. One main difference to XCP is that in RCP the routers offer the same rate to all flows. When a path is congested, these methods allow short flows to finish more quickly. Quick-Start, XCP and RCP all suffer from deployment difficulties; not just the need to update routers and switches, but also the fact that an end-system cannot jump to the agreed rate in case it is traversing a lower layer bottleneck that does not support the mechanism.

The discussed proposals for rapidly sensing capacity mainly target short to medium size flows where they can significantly reduce the number of transmission rounds required. In practice IW10 can bring a saving of up to four RTTs [246] and the other schemes have potential for even larger savings. Longer flows see less proportional benefit, because the start-up phase is of lesser significance for long flows. IW10, Jump Start, Quick Start and RCP have been evaluated and compared by Scharf in [250]. He found an increased initial congestion window to be the most promising of these proposals, since Jump Start seems to be too aggressive, while Quick Start and RCP require modifications to the entire network to be efficiently deployed, and have not seen any practical use. Scharf also found an increased initial congestion window to be a reasonable improvement, since it is easy to deploy and it should not endanger network stability. Still, an increased initial congestion window can only offer a limited performance improvement in relation to what is possible, and larger IW values are currently not thought safe for more general deployment.

D. Collateral Damage

The Internet path between endpoints is a shared resource, that can introduce a common bottleneck in which traffic from one or more flows can influence the delay and loss experienced by another flow. One responsibility of the transport layer is to promote good sharing of this path, and the IETF therefore requires all traffic to use some form of congestion control algorithm [266]. Moreover, applications vary significantly in both the characteristics of the data they would like to send and their expectations of the network conditions (i.e., throughput, RTT, loss rate, latency, etc). This section describes methods to mitigate inducing unnecessary latency amongst flows. It initially focuses on transport for Scavenger applications, followed by bursty TCP applications, and finally how to avoid slow start overshoot.

The methods described below in the context of TCP and a Scavenger Class can be applied to any transport protocol, including SCTP, DCCP and other protocols based on UDP. Although these methods can help avoid an unnecessary increase in application latency, they are not appropriate for all traffic. Applications that choose to use UDP in preference to TCP, SCTP or DCCP do not necessarily react to loss or delay on the same time-scale as TCP would. Some UDP-based applications do not react at all [149], or react only after many RTTs (e.g., circuit-emulation service [267]).

1) Low Priority Congestion Control: Most bulk traffic (i.e., from applications that need to transmit continuously for periods of many RTTs) use TCP. TCP probes for network capacity, by continuously trying to grow the congestion window to the point at which congestion is detected by loss or timeout. This has the side effect that a bulk TCP flow builds a queue at the bottleneck router, which causes flows sharing this bottleneck to experience increased latency [268].

One approach to reduce the impact of TCP capacity probing, is to use increased (queuing) delay as a metric for detecting capacity, see Section IV-F7. Delay-based methods could exploit this to build a Scavenger Class application. This class of application voluntarily accesses network resources with lower priority than other traffic, preferring to quickly yield capacity to more important traffic when multiple flows compete (see Fig. 19). For example, transport designed for bulk background peer-to-peer transfer or aggressive pre-fetching that desires to capitalize on excess capacity when available. Work in this

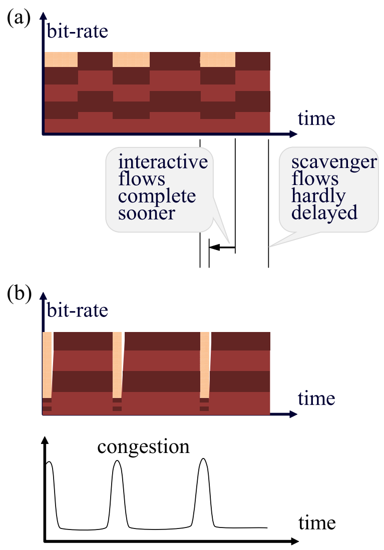

(a)
bit-rate
time
interactive flows complete sooner
scavenger flows hardly delayed
(b)
bit-rate
time
congestion
time

Fig. 19. Scavenger congestion control. (a) Equal rates. (b) Scavenger flows yield to congestion.

domain includes TCP Nice [269], which combines delay-based inference of congestion with an aggressive reduction of the send rate, allowing also a send rate below one packet per RTT. TCP Low Priority (TCP-LP [270]) proposed a similar delay-based approach, showing that using this for background transfers instead of TCP can reduce the response time of competing Web connections by around 80%.

In 2012 the IETF defined Low Extra Delay Background Transport (LEDBAT [213]), an experimental method to support Scavenger applications. In the absence of competing flows, a LEDBAT flow is intended to sustain a rate as high as possible while keeping the queuing delay along the end-to-end path close to a target value. The method estimates the *base* delay (i.e., the minimum delay in the absence of queuing), and then adapts a congestion window in proportion to the difference between the measured queuing delay and a predefined target (with 25 and 100 ms as typical values). This is intended to allow LEDBAT to conservatively decrease its sending rate in the presence of queuing delay.

LEDBAT and similar techniques have been widely deployed in peer-to-peer clients, and have been adopted by some OS vendors to deliver software updates. However, some authors [271]–[273] have reported potential issues that call into question its intended Scavenger behavior, especially its impact on latency-sensitive applications like web browsing [273], [274]. Some of the difficulties in obtaining a reliable delay-based congestion signal are shared with TCP Nice and TCP-LP, with similar potential issues also applying for these protocols.

Recent work by Gong *et al.* [275] has also shown a potentially negative interaction between AQM and low priority congestion control. The AQM mechanisms in effect remove the prioritization between flows that the low priority congestion control is trying to achieve. Transport and network methods need to therefore be designed to complement each other [149] (see Section IV-F6).

2) Congestion Window Validation: Some classes of applications generate data at a variable rate determined by the application itself rather than trying to send as fast as possible, for example remote storage and real-time transmission. In the context of TCP, this corresponds to a flow that needs to send appreciable volumes of data, but does not need to maximize throughput as in bulk transfers.

Variable rate applications that use TCP can significantly impact the latency of other flows. This is because a TCP sender can accumulate a large congestion window, even if the application is not generating sufficient data to probe for a higher rate. If the application then increases the rate, this can inject a (line-rate) burst of data into the network. Congestion Window Validation (CWV [276]) attempted to solve this by limiting the congestion window to the actual used value.

Experience subsequently showed that while CWV addresses a valid need, the algorithm in [276] can add latency to real-time applications [277]. This resulted in little use of CWV, so recent IETF activity is updating CWV to permit bursty applications to accumulate a larger, but controlled, congestion window coupled with a more aggressive reduction in rate when congestion is detected using the unvalidated window [278].

The new CWV method seeks to satisfy the capacity requirements of variable-rate latency-sensitive applications, and to reduce latency for flows that share a bottleneck with bursty TCP applications [279]. It may also improve performance for variable-rate TCP applications that encounter a network bottleneck or a change in their network path characteristics.

3) Avoiding Slow Start Overshoot: Excessive bursts of data may also be inserted into the network during connection start-up (Section V-C2). The exponential increase of the sending rate during slow start may result in severe overshoot of the available capacity. This can not only result in packet loss for a flow itself, but also causes packet loss and increased delay for other competing flows. Avoiding (or at least limiting) slow start overshoot can thus help reduce latency.

Hybrid slow start (HyStart [245]) is one solution targeting slow start overshoot for bulk flows. HyStart does not change TCP's basic slow-start algorithm, but attempts to exit slow start before a severe overshoot occurs. It applies two independent heuristics to find a suitable exit point. First, it monitors when the duration of each whole acknowledgement train approaches the RTT, which indicates that the congestion window is large enough to utilize the available path capacity. Second, it monitors increases in the delay for the acknowledgements sent at the beginning of each round, which indicate that queuing delays are building along the path. HyStart is implemented as part of the CUBIC congestion control module used as a default in Linux.

Earlier work in this area includes the Limited Slow Start mechanism defined in [280]. This adds a maximum value for the slow start threshold, `max_ssthresh`. Once the congestion window grows beyond `max_ssthresh`, the increase is limited to `max_ssthresh`/2 segments per round. Selecting a suitable value for `max_ssthresh` is, however, difficult. Paced Start [281] monitors the queuing delay that a buffer adds between packets when sent in trains during TCP slow-start and paces the packets sent in subsequent rounds. CapStart [282] uses a combination of limited slow start and classic slow start. Limited slow start is used unless the sender measures that the bottleneck is probably at the sender and not in the network, in which case it reverts to classic slow-start. Because the `ssthresh` value determines the transition point from slow start to congestion avoidance, it can be dynamically tuned to avoid overshoot. Methods for tuning the `ssthresh` value based on estimates of the available bandwidth are for example explored in [283] and [284]. The challenge in all cases is that limiting the sending rate or leaving slow start too early will increase the time required to reach the available capacity.

VI. Intra-End-Host Delays

Host latency results from how applications are designed and optimized to locate, request and process data. This depends on the operating system and is constrained by the hardware technology of the host. This section analyzes delays due to processes within the host, as opposed to processes in-between hosts. Buffering delay and head-of-line delays are associated to protocol stacks and how they are designed and operate at the end points, and are detailed in the next subsections. Moreover, operating system delays include delays associated not only to the software, but also to the hardware of the host. In this framework, latency depends on the fundamental design and architecture of the systems.

A. Transport Protocol Stack Buffering

The network stack needs to buffer data passing between the network and application(s). Host stacks provide buffering at both the sender and receiver, since generally applications execute outside the kernel where protocol processing is normally performed. Excessive host buffering not only adds to end-to-end latency, it also can result in excessive resource consumption. A straightforward mitigation is to limit the maximum queue size, although in practice there are many places in the stack where buffering may be required. Still, there is evidence that operating systems are being updated to limit the effects of bufferbloat, with the bufferbloat project [285] as a strong driver for this work.

Semke *et al.* [286] showed that tuning the send socket depth to $2 \times$ `cwnd` could avoid excessive buffering for bulk TCP flows without impacting throughput. TCP small queues (TSQ) also modified the Linux kernel [287] to limit the default network socket to 128 KB, to avoid sender latency. Moreover, Goel *et al.* [288] showed a trade-off between the depth of socket input buffering and the TCP throughput that could be used to optimize performance when an application wishes to trade the maximum throughput for reduced latency. In real-time applications (e.g., using the Real-Time Protocol, RTP), a common issue is that network latency can cause data to arrive after a deadline, and is then of no (or limited) value; excessive input buffering adds to this latency. Byte Queue Limits (BQL [153]) has been added to Linux to dynamically minimize the hardware transmit buffer, thus pushing any larger queue higher up the network stack where AQM can be applied (see Section IV-F2).

Latency can also be introduced when data needs to be copied between a buffer and application, kernel, or device memory. A Zero-copy technique avoids redundant copying between intermediate buffers and associated context switches [289]. This

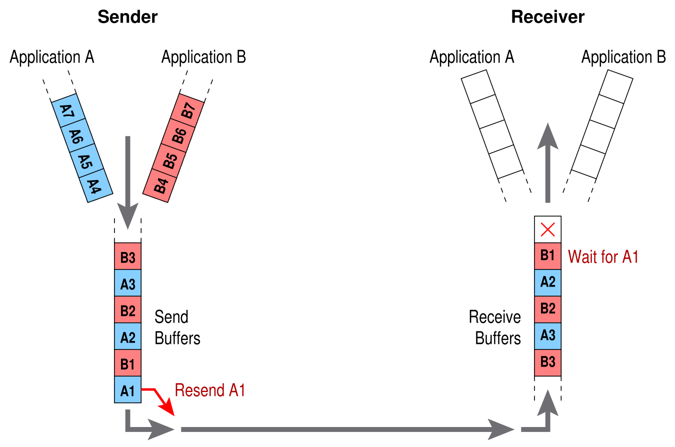


Fig. 20. Head-of-Line blocking delay, due to loss of packet A1.

can significantly benefit both bulk transfer over high-speed networks [290] and latency-sensitive real-time applications [291].

B. Transport Head-of-Line (HOL) Blocking

TCP's stream-based design requires the socket API to sequentially deliver data to the receiver. This can cause head-of-line (HOL) blocking with added delay when packets are lost or reordered during transmission.

Reliable ordered delivery is not a requirement for best-effort, datagram-based protocols based on UDP, or for DCCP. Datagram-based protocols do not require significant receive buffering, leaving any re-ordering decisions to the application, where these may be kept to a minimum for latency-sensitive applications. When data is framed as separate messages (e.g., application-layer framing), it may be possible for an application to skip data that is no longer needed or avoid waiting for data that was not received. This is provided by DCCP and by PR-SCTP [94] which permits a stream-based partial reliability (see also Section III-C1). DCCP and UDP-Lite can also deliver data when the checksum fails on all or parts of a packet, potentially eliminating HOL blocking delay.

SCTP can also reliably deliver datagrams when they do not arrive in the order in which they were sent. This avoids HOL blocking delay for applications that can accept out-of-order data, or when ordered data streams are multiplexed together such that the ordering of datagrams on the wire breaks only *between*, not within the streams. Fig. 20 shows two application streams, A and B, sent over a single TCP connection, where ordering due to the loss of A1 causes unnecessary delay for application B. SCTP's multi-streaming provides a form of multiplexing without this delay. This matches SCTP's goal to support latency-sensitive transaction processing applications multiplexed over a common transport entity.

The solutions described in Section III-C for reducing packet loss recovery delays can also all help reduce or eliminate HOL blocking delay caused by packet loss. In particular, ECN and FEC can remove or hide the losses that are the underlying cause of the blocking, respectively (Section III-C). ECN only removes congestive losses, whereas FEC hides both congestive and transmission losses. FEC can potentially also help reduce HOL blocking delay due to reordering.

The current *sockets* application programming interface (API) provides the stack with very little information about the data being sent. Many of the issues with protocol buffering could be eliminated if the sender socket interface were richer and provided additional information about the relative value and expected timeliness required for transmission [292], [293]. A richer API could also ease migration of services to other transports more tailored to latency-sensitive applications (e.g., DCCP and SCTP) [294].

C. Operating System Delays

The operating system must orchestrate and shield the diversity and complexity of hardware from software. Application software uses APIs to interact with the operating system, network stack and other applications. The host hardware technology—more specifically, how the CPU, memory and I/O devices (e.g., NICs) are designed to exchange and process data—has a fundamental impact on the latency experienced by applications [295].

Techniques such as parallelization, pipelining and zero-copy can increase performance and reduce latency. Many forms of parallelism can reduce latency [296], [297]. Bit-level parallelism improves the processor, buses or memory word sizes. Instruction-level parallelism utilizes multi-core or instruction pipelining. Data parallelism takes advantage of the possibility of multiple processors accessing part of the same data concurrently, while task parallelism addresses independent tasks that can be done concurrently. Parallelizing is not always straightforward; data and task synchronization in a multi-threaded environment can lead to the so-called "race condition" problem [298]. Multithreaded programming addressed race conditions using locks, mutexes and semaphores, however detecting race conditions is an NP-hard problem [299].

Addressing latency in the host is not only a matter of faster hardware, it also implies addressing the architecture of the system. Chip manufacturers already optimize and integrate frame control, scheduler, memory and data path logic in the silicon fabric [300]. Network-On-Chip is an active field of research [301], [302], which proposes to integrate network and CPU functions on the same die [303], [304]. Rumble *et al.* [305] argue that a system capable of achieving remote procedure calls with 5–10 μs delays for data centre applications is possible, with a long-term objective of achieving 1 μs delays. Rumble *et al.* further urge silicon, operating system and software developers to work together to achieve this long term goal.

VII. Composite Solutions

Many mechanisms bundle techniques that address multiple sources of delay. This section illustrates this by describing three such mechanisms: A) SPDY—an application-layer web protocol mechanism, B) QUIC—an experimental stream-oriented multiplexing protocol based on UDP, and C) WAN accelerators—edge devices that use a variety of techniques to accelerate an enterprise's WAN connection.

A. SPDY

SPDY [306] is an application-layer web protocol that reuses HTTP's semantics. As such, it retains all features including cookies, ETags and Content-Encoding negotiations. SPDY only replaces the manner in which data is written to the network. The purpose of this is to reduce page load time. It does this by introducing the following mechanisms:

- *Multiplexing*: A framing layer multiplexes streams over a single connection, removing the need to establish separate TCP connections for transferring different page resources.
- *Compression*: All header data is compressed to reduce the overhead of multiple related requests; [306] reports a reduction of ∼85% in HTTP header sizes, and a reduction in web-page download times due to smaller headers that can be as high as 1 s in a low-bandwidth scenario.
- *Universal encryption*: SPDY is negotiated over SSL/TLS (Section III-B1) and thus operates exclusively over a secure channel. This adds delay to complete the key exchange in the SSL/TLS Handshake Protocol.
- *Server Push/Hint*: Servers may proactively push resources to clients (e.g., scripts and images that might be required in the future); alternatively, SPDY can send hints advising clients to pre-fetch content (see Section II-C3). A downside of the drive to reduce web browser latency is that, with this feature, SPDY sessions can potentially transfer much more content that traditional methods, impacting the latency of traffic that shares a network bottleneck.
- *Content prioritization*: A client can specify the preferred order in which resources should be transferred.

SPDY consists of two components. The first provides framing of data, thereby allowing features like compression and multiplexing. The framing layer works on top of secure (SSL/TLS) persistent TCP connections that are kept alive as long as the corresponding web pages are open. Clients and servers exchange *control* and *data* frames, both of which contain an eight-byte header. Control frames are used for carrying connection management signals and configuration options, while data frames carry HTTP requests and responses. The second component maps HTTP communication into SPDY data frames. Multiple logical HTTP streams can be multiplexed using interleaved data frames over a single TCP connection.

A recent study [307] found that Server Push is currently largely unsupported, leaving most of the potential latency gains of SPDY to multiplexing. Agreeing with other earlier studies, the authors of [307] find it hard to document a consistent advantage or disadvantage except for low-bandwidth/high-delay environments. Notably, while SPDY reduces HOL delay at the application layer (e.g., a stream does not have to wait to be transmitted until another one that consists of dynamically generated content is ready), multiplexing over TCP still gives SPDY the RTT-timescale HOL blocking delay shown in Fig. 20. This may be addressed by QUIC (Section VII-B).

B. QUIC

QUIC (Quick UDP Internet Connections) is a project by Google to design a new stream-oriented multiplexing protocol for the next-generation web. The protocol is intended to provide low latency for services equivalent to TCP, but built upon UDP. UDP was chosen because it offers immediate deployment and provides a high connectivity success rate through middleboxes. QUIC seeks to combine a carefully selected collection of layer-4 techniques to decrease application latency for SPDY and HTTP2.0 [308].

The main features of QUIC that affect latency are:

1) Avoiding the head-of-line blocking that would be incurred by streams multiplexed over TCP. Being based on UDP, QUIC can also avoid the initial handshake TCP would have had to perform to set up the connection.
2) Intrinsic TLS-like security that seeks to eliminate the TLS handshaking delay [309].
3) Pacing to reduce loss due to bursts from congestion window cycles.
4) FEC to reduce the delays caused by the need for retransmissions.

Initially QUIC will be explored within the open source Chromium project [310] and to experiment with mechanisms that work across multiple protocol layers. In the long term, this could result in an alternative protocol to TCP—although experience of using and understanding the new mechanisms could also result in updates to TCP itself rather than evolving a new standard protocol. As QUIC is still in a prototype phase and several of the sub-mechanisms are immature at the time of writing [311], it is hard at this time to estimate the potential gain and possible effects from the combinations of mechanisms deployed in QUIC.

C. WAN Accelerators

WAN accelerators combine many of the component techniques discussed in this paper, many of which were pioneered by WAN accelerator vendors. WAN accelerators are typically deployed by an enterprise that uses a virtual private network (VPN) to interconnect branch site(s) and a centralized data centre(s) or headquarters site(s). They are generally targeted for deployment at the border of each site, to cover the segment of the path over the wide-area network between sites. To some extent they can be thought of as a way to speed up deployment of techniques that ought eventually to be implemented in the stack of every end-system. However, they also exploit aggregate knowledge that is not easily available in every end-system.

1) Structural Arrangements for WAN Acceleration: Traffic typically traverses a pair of WAN accelerators, one at the egress of the sending site, and the other at the ingress to the receiving site. Variants of this arrangement exist. For instance in a mobile workforce scenario the 'branch site' end of the pair of accelerators may be software installed on each employee's mobile device, e.g., Riverbed's Steelhead Mobile Client [312].

Ipanema's products are unusual in that they do not just act in pairs; the set of boxes around the edges of a VPN over a meshed wide-area network communicate with each other to optimize traffic leaving the mesh from any one egress (the only published explanation is in Ipanema's patents [313]–[315], but [316, Fig. 4] gives a high level overview). This approach can be applied one-ended, so that all traffic is still routed via a WAN accelerator at the data centre end, but branches do not need to deploy a device.

The benefits of WAN acceleration are starting to be eroded with the trend towards local Internet break-out at each branch site. This so-called 'hybrid connectivity' uses the Internet to reach public cloud data centers while the VPN provides connectivity to private resources as well as to the same public data centers when Internet quality is poor. Ipanema partially addresses this shift with a product that sits at a branch site and automatically selects between the Internet and the VPN for each user flow [317]. However, WAN accelerators cannot be fully effective unless they control all the traffic passing through the devices.

2) WAN Acceleration Features: Here we focus exclusively on the features of WAN accelerators that address latency, ignoring techniques that address bandwidth efficiency (e.g., correctly scaling TCP's receive-window or using high-speed TCP). Below is a list of latency techniques that WAN accelerators employ. Each specific make and model of WAN accelerator will provide only a subset of these. Where the details may be of interest, some are explained further afterwards.

Techniques may be either generic, transport-specific (e.g., applicable solely to TCP), or application-specific:

- Generic
  - Eliminating duplicate data transfers (loosely related to Section II-C1 on caching and to Section V-B on redundant information);
  - QoS marking and enforcement by application (see Section IV-F4);
- Transport
  - Connection Pooling (related to Section III-A4 on pipelining);
  - SSL acceleration (loosely related to Section III-B1);
  - Rapid filling of available capacity (related to Section V-C2 on rapidly sensing available capacity);
- Application
  - Cache pre-fetching (see Section II-C1);
  - Reduce round trips of important but inefficient application layer protocols.

According to the Riverbed Optimization System (RiOS v6.1) Technology Overview [312], RiOS uses many of the techniques listed above. Riverbed's data de-duplication is a compression technique that replaces sequences of bytes with index codes. It works at a minimum granularity of about 100 B, but the codes work hierarchically, so they can refer to a large file with only small differences from an otherwise similar file transferred earlier. This de-duplication approach is independent of the application protocol, unlike caching (Section II-C1), which needs to understand the objects and identifiers that an application layer protocol uses. It can therefore compress e-mail attachments, remote print serving, java applets, back-ups, etc.

At the transport level, Riverbed's accelerators maintain a pool of pre-opened connections between the ends of the WAN segment, to deal with TCP's handshaking round trips in advance, that would otherwise slow down short-lived connections. Riverbed avoids aggregating many TCP connections into one TCP tunnel, which can lead to classic TCP-over-TCP performance problems and to packet fragmentation due to tunnel header overhead. Instead each WAN accelerator device acts as a TCP proxy by intercepting each TCP connection and mapping it into another TCP connection over the wide area, which is in turn mapped into a third TCP connection between the remote WAN accelerator and the remote end-point.

In contrast, at the transport-layer, current Ipanema WAN accelerators focus on minimizing TCP's slow-start latency. They continuously co-ordinate with each other to determine the capacity each WAN segment should consume into each branch. Then each TCP flow immediately opens up its window to fully utilize this allocated capacity (bandwidth usage is also optimized as part of this process, but that is outside our current scope).

It has been proposed that WAN accelerators use the RSVP reservation protocol to ensure that capacity is available on interior links as well as on edge devices [318], so that immediately opening a full window does not overload core network links. However, most vendors take the pragmatic approach of solely managing access link bottlenecks into and out of the WAN.

Some vendors concentrate on a limited sub-set of the techniques; for instance, Infinita focuses on reducing the time for TCP slow-start as well as using high-speed TCP for the continuing connection [319]. Nonetheless, eliminating duplication is common in most solutions, because it often gives the greatest gains (e.g., Riverbed claims it typically reduces flow sizes by 65% to 95%).

It is also very popular for WAN accelerators to remove chatty handshaking rounds in application-layer protocols that were originally designed for LAN rather than WAN environments. The classic example is Microsoft Windows networked file system (CIFS), which used to be very inefficient before it was redesigned for wide area networks [320]. Streamlining is also popular for other common enterprise applications such as Microsoft SQL Server databases (the TDS protocol); Lotus Notes; and the Oracle e-Business application suites. However, standardized protocols also offer considerable scope for proprietary latency improvements, e.g., Network file system (NFS); server-based email (MAPI); and HTTP/HTTPS. Many application-specific WAN acceleration features enhance old versions of the protocols, and become redundant as the application-layer protocols evolve to efficiently support wide-area networking.

Application-specific knowledge can also be used to pre-fetch data. For instance, caching at a branch under the assumption that several people will probably download the same corporate email or objects hyperlinked from popular Web pages.

Some WAN accelerators include other miscellaneous techniques, such as using FEC to mitigate packet losses (see Section III-C), enforcing bandwidth limits, or placing limits on the number of connections (to protect equipment from flow-state exhaustion).

3) Performance Enhancing Proxies (PEPs): There are also sets of performance accelerators aimed at links with specific characteristics, such as intermittent connectivity, variable bandwidth on demand, high latency, high loss, etc. These devices act similarly to WAN accelerators and often provide similar functions, but can also leverage cross-layer features—such as information about the network held in a service subscriber database, access to radio resource management functions to accelerate transmission of key data, and the ability to setup and manage lower layer bearers with specific capabilities. Such devices are widely deployed to support wireless, cellular and satellite services, especially to manage application performance in the face of high or unpredictable delay or variable loss (see [321] for further information).

VIII. Classifying Solutions in Different Ways

To determine a sound structure for this survey, a few alternative classification structures were considered. We arranged all the techniques using each scheme to test which one would both encompass all the material and lead to fewest overlaps.

An alternate way that we explored for structuring this survey was based on the *phase(s) of a communication session* that each technique addresses. In principle, any exchange of data between endpoints can be regarded as being divided in (up to) three phases over time:

1) Session startup.
2) "Getting up to speed" (GUTS), i.e., assessing a safe sending rate for the path by progressively increasing the sending rate.
3) Data transfer.

Taking a long-lived, TCP-based bulk transfer as an example, (1) corresponds to the three-way handshake, (2) to the initial slow-start phase, (3) and to transfer of most of the data after the initial slow-start transient (with the sender mainly in congestion avoidance). Of course, this mental model does not necessarily apply as-is to any arbitrary data exchange, nor are all phases always non-overlapping. For instance, for short-lived TCP flows transfer of all data (phase (3)) may well happen without the sender ever exiting slow-start (phase (2)). Nonetheless, in spite of such potential ambiguities, the model is convenient since some latency-reduction techniques focus on specific phases of the lifetime of a session.

Table II illustrates the way in which the types of techniques explored in the previous sections map into different phases of a session. An 'x' symbol denotes that a given technique can likely bring latency-reduction benefits in the corresponding phase, whereas an '?' symbol indicates that benefits are not clear-cut and may depend on several factors, or may involve tradeoffs; the reader is referred to the relevant sections for details. Note that many of these techniques are applicable to, or have an impact on, more than one of the three phases described above, implying that such a classification would be of limited use.

Another way of structuring a taxonomy of techniques that can reduce latency would be to map the techniques to the layers of the OSI model. However, we discarded this obvious, easy classification scheme since it did not offer much insight into the problems each solution addresses; also, we found that it obscured the fact that some types of methods may be applied at multiple different layers of the stack, or cannot be simply mapped to a single layer because they may require the concerted actions of elements in more than one layer.

For instance, techniques such as TFO for faster opening of TCP connections (Section III-A3) are clearly located at the transport layer. However, TCP Early Retransmit (Section III-C2) also sits clearly in the transport layer, yet it addresses very different issues than TFO. Methods like compression of protocol headers may be applied to application-layer protocols (as, e.g., SPDY does, see Section VII-A) or to network- and transport-layer headers (as, e.g., ROHC does, see Section V-B). Yet they tackle a similar issue—protocol header overhead. Further, some other techniques, like Active Queue Management (Section IV-F6), involve the interaction of entities located at more than one layer of the stack (e.g., the transport protocol and a buffering system in a lower layer, in the case of AQM).

Even within the structure we adopted, the top level sources of delay to use were not obvious (and they are different from those proposed by [322]). For instance, we do not have a section for delay due to interaction between an endpoint and the network, even though we have one for interaction between endpoints (Section III). A number of the delays are certainly exacerbated by a lack of explicit interaction between endpoint and network in the Internet architecture, e.g., those to do with sensing available capacity in Sections V-C2 and V-D. Nonetheless, it did not seem appropriate to identify this interaction as a source of delay, when it is actually the *absence* of this interaction that causes delay. Therefore, instead we chose to classify these delays by the underlying source of delay, e.g., attempting to sense capacity.

If interaction between endpoint and network had been one of our top level classifications, the following techniques would have fallen within it:

- Introduction of explicit signals:
  - — Classic ECN (Section III-C4)
  - — Instant ECN, e.g., DCTCP (Section IV-F6a)
  - — Virtual queue ECN, e.g., HULL (Section IV-F6a)
  - — Lower priority for probe packets, e.g., Fast-Start, TCP-Peach (Section V-C2)
  - — Explicit rate signaling, e.g., Quick Start, XCP, RCP (Section IV-C2)
- Working round lack of explicit signals:
  - — Delay-based congestion control (Section IV-F7c)
  - — Chirping to detect buffer capacity (Section V-C2)
  - — Increasing TCP's initial window (Section V-C2)
  - — Congestion window validation (Section V-D2)
  - — Detecting slow-start overshoot using delay (Section V-D3)

IX. Gain vs. Deployability

Having surveyed a wide range of techniques, we now aim to summarize the merits of the main types of technique. Our primary approach will be to visualize the gain in performance of each technique against the potential difficulties and cost foreseen in getting it deployed—gain vs. pain.

A. Gain

Quantifying the benefit of each technique requires consensus on a figure of merit. We decided on **percent reduction in session completion time**

$$= 100\% - \frac{\text{session-completion-time}}{\text{original-session-completion-time}}.$$

In general, we take the baseline for original session completion time as the state of the art technology used in production systems at the time of writing (2014).

TABLE II
ALTERNATE TAXONOMY, BASED ON THE TYPICAL PHASES OF A COMMUNICATION SESSION

		Phase(s) of a session addressed		
Type of technique	**Discussed in Section**	**Startup**	**GUTS**	**Transfer**
Shorter-latency routes	II-A	x	x	x
DNS pre-fetching	II-B	x		
Network Proxies and caches	II-C1	x	x	x
Client caches	II-C2	x	x	x
Data prediction and latency-hiding	II-C3	x	x	x
Better service placement	II-D	x	x	x
Parallel option negotiation	III-A1	x		
Reducing NAT setup delay	III-A2	x		
Fast opening of TCP connections	III-A3	x		
Application pipelining	III-A4	x	x	
Path MTU discovery	III-A5	x	x	x
Faster transport security negotiation	III-B1	x		
Building encryption into TCP	III-B2	x		
Bootstrapping security from the DNS	III-B3	x		
Trading reliability for lower latency	III-C1		x	x
Reducing packet-loss detection times	III-C2		x	x
Adding redundancy to TCP	III-C3		x	x
Explicit congestion notification	III-C4	x	x	x
Enhanced Nagle	III-D			x
Straighter cable paths	IV-A1	x	x	x
Higher signal velocity	IV-A2	x	x	x
Combining higher signal velocity and straighter routes	IV-A3	x	x	x
Optimizing MAC-layer protocols	IV-B	x	x	x
Reducing serialization delay	IV-C	x	x	x
Improving link error control or channel quality	IV-D	x	x	x
Lower switch processing delays	IV-E	x	x	x
Flow and circuit scheduling	IV-F1	?	x	x
Limiting MAC-layer buffering and flow control	IV-F2	x	x	x
Limiting IP-layer buffering	IV-F3	x	x	x
Packet scheduling	IV-F4	x	x	x
Traffic shaping and policing	IV-F5	x	x	x
Active queue management	IV-F6	x	x	x
Transport-based queue control	IV-F7		x	x
Leveraging multiple links / interfaces to increase capacity	V-A	x		x
Avoiding transmitting redundant information	V-B			x
More aggressive congestion control	V-C1			x
Rapidly sensing available capacity	V-C2		x	
Minimizing collateral damage	V-D	x	x	x
Limiting buffering in the host's network stack	VI-A	x	x	x
Avoiding transport HOL latency	VI-B			x
Reducing Operating system delays	VI-C	x	x	x
SPDY	VII-A	x	?	x
QUIC	VII-B	x		x
WAN accelerators	VII-C	x	x	x

'Session' is a deliberately general concept that can be stretched to mean a message, a connection or a set of connections that fulfill a task. This allows us to compare techniques that address a range of interactions, as long as the content of the session is the same in the before and after scenarios (denominator and numerator):

- a one-way end-to-end event notification (message), e.g., a price update;
- a two-way exchange (connection) including connection set-up and optionally security set-up, e.g., retrieval of a simple Web page;
- a set of exchanges of information (session), e.g., retrieval of a geographical map identifying hardware shops in a locality.

'Completion time' was chosen to focus on the whole of a task, but can also be used to measure the time to complete

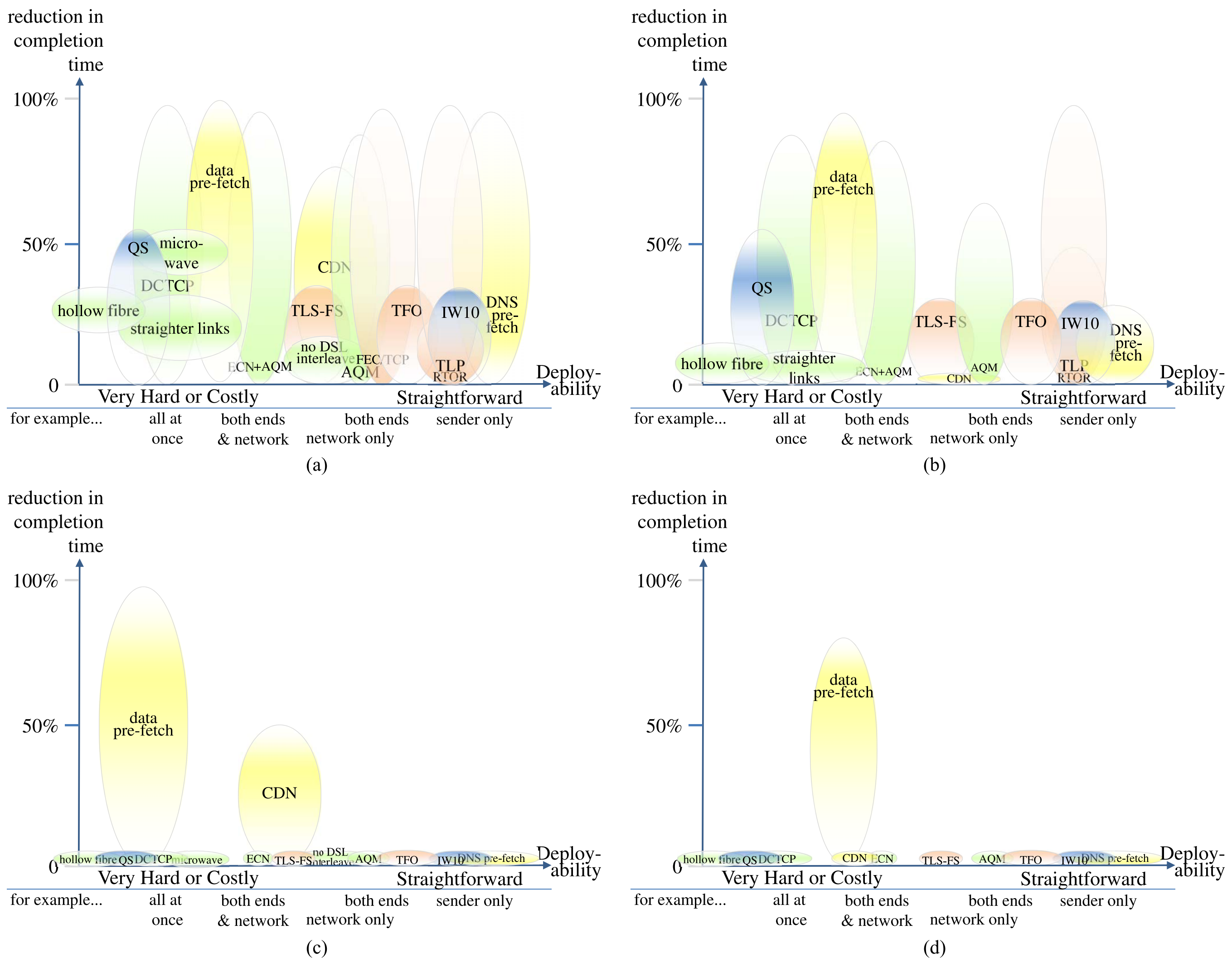


Fig. 21. Bubble plots of rough latency gains against ease of deployment for a selection of techniques. The heights of the captions of each bubble represent typical values, and the vertical extent of the bubble represents variance. See Section IX-D for expansion of abbreviations and commentary on each technique. Some techniques discussed in the commentary are not included in the diagram because their latency reduction is too scenario-specific. (a) Small session (∼20 kB) flows over WAN. (b) Small session (∼20 kB) flows over LAN. (c) Large session (> 2 MB) flows over WAN. (d) Large session (> 2 MB) flows over LAN.

the component steps in a process. The term latency is often associated with the time spent getting started before the body of a task can start, and completion time can measure completion of the start-up phase. It is not always natural to stretch this to the extreme. For instance the latency of a streaming video is the delay between requesting it and the first frame being played-out, but ‘completion time’ is not a good term for this. Nonetheless, as long as it is qualified by what is being completed, completion time is a useful general metric.

One disadvantage of using reduction in completion time is that all the techniques with the most startling results bunch up at just under 100%. To solve this, a possible alternative metric would have been **speedup factor**

$$= \frac{\text{original-session-completion-time}}{\text{session-completion-time}}.$$

However, then the majority of reasonable techniques would bunch around 1–1.5. Given few techniques give exceptional improvements in performance and most give only moderate gains, we use percent reduction in completion time, except if we need to bring out the distinctions between those with exceptional improvements.

B. Pain

The difficulty foreseen in deploying a technology is necessarily subjective. We arrange the techniques on a rough scale from ‘Very Hard or Costly’ to ‘Straightforward.’ The scale does not extend to ‘Easy’ because it is debatable whether deployment is ever easy in a complex system like the Internet.

In general, techniques that offer immediate performance gain when unilaterally deployed on an end-system are placed at the ‘Straightforward’ end of the spectrum. On the other hand, techniques that require the sender, receiver and all network elements on the path between them to be changed before there is any benefit are considered very hard to deploy, particularly if all parties have to change at once. Nonetheless, we have still categorized some techniques that are unilaterally deployable as 'Very Hard or Costly' because of their prohibitive cost (e.g., replacing long-distance fiber links with point-to-point microwave links).

C. Caveats and Scenarios

Fig. 21 arranges a small selection of the techniques in the full survey as bubble plots. The vertical axis represents reduction in session completion time and ease of deployment is shown on the horizontal. Bubble diagrams are generally approximate, which suits the rough nature of the data being presented.

In general, bubbles higher and to the right are better. Nonetheless, it is important not to ignore techniques on the left, because subsequent research might succeed in making a technique easier to deploy.

The vertical positioning of each bubble's caption represents the typical reduction to be expected, while the vertical extent of each bubble roughly represents the likely variance of the latency reduction. We admit that variance of reduction in completion time is not an intuitive metric, because it usually has more to do with variance of completion time before the solution is applied, not variability in how well the solution reduces completion time. Wider bubbles merely show that a technique's deployability is less certain.

Such a simple visualization cannot hope to summarize all the factors involved, which can only be appreciated by looking up the relevant techniques in the body of this survey and following up the references if necessary. This interim visualization alone should not be used to prioritize work, nor to identify gaps in the solution space, because:

- it only includes a small selection of the techniques in the present survey;
- quantification of latency reduction often relies on evidence from the promoters of a particular technique without independent verification;
- with such a wide variety of techniques, it is hard to ensure that the baseline cases are all comparable before calculating the reduction in latency of each technique;
- the reduction in latency of any technique depends on the prevailing scenario—the levels of transmission and congestion losses, the bottleneck capacity, the degree of multiplexing, the patterns of foreground and background traffic arrivals, the topology, etc.;

However, we felt a selection of ball-park figures would be more useful than no quantification at all.

Despite the numerous parameters above that could characterize a scenario, two parameters in particular strongly affect the outcome in nearly all cases:

- the amount of data transferred ('session-size');
- how far apart the end-points are (or were originally), e.g., WAN, LAN.

It should be sufficient to visualize just two cases for each of these two dimensions, leading to the 2×2 matrix of cases shown. We assume about 20 kB of data for the small session-size, and we will take a large session-size to mean more than two orders of magnitude greater (> 2 MB). Fig. 21 illustrates the case of a WAN ($\sim$200 ms) and a LAN ($\sim$2 ms RTT).

D. Commentary

The 'gain and pain' of the techniques selected for Fig. 21 is discussed below, organized by the main sources of delay that it addresses. The bubbles are colored to match the main sources of delay in Fig. 2.

1) Structural:

Optimize routes (Section II-A):

The scope for reducing path latency by optimizing the route is distinctly scenario-specific. Chetty *et al.* [323] measured the latency from Johannesburg in South Africa to various servers around the world and compared it with the distance 'as the crow flies.' Dividing the measured latency by the time it would take for light to traverse the direct distance in glass gives the stretch factor introduced by the Internet routing system, further compounded by circuitous cable runs, i.e., stretch=measured_latency/ideal_latency. The stretch factor to servers in the southern hemisphere or equatorial regions was always more than 2.5. In contrast, the stretch factor to northern hemisphere servers was typically about 1.3. The explanation was that all routes traversed northern hemisphere Internet exchanges, even when the other end was also in Africa, such as Nairobi in Kenya, which exhibited the highest stretch of 6.9. Reducing the worst stretch (6.9) to the best (1.25) would represent a reduction in delay of $1 - 1.25/6.9 \approx 82\%$.

In contrast, optimizing route latency within a network in a more mature market offers diminishing returns. For instance, an unpublished study conducted on a European national core network found that edge-to-edge latency could be reduced by about 3.2% on average by rearranging the OSPF link weights. This gain would be smaller when translated into a reduction in end-to-end delay, because the network core is only a proportion of the end-to-end path, and there is little scope for route optimization in the access portion. The same study showed that further reducing delay on every route to its absolute minimum ('as the crow flies') would lead to an average reduction in delay of 79%—nearly as bad as the worst case in the Johannesburg study. This is perhaps not surprising, since the relatively small absolute reductions in delay that can be achieved at a national scale would not be worth the investment in thousands of routes between pairs of towns, at least not in comparison to the greater potential gains in absolute terms between a few major international centers.

DNS (Domain Name System) pre-fetch (Section II-B):

DNS pre-fetching will save on average 250 ms for a Web user once a link is followed [324]. A survey on DNS lookups in Jung *et al.* [17] shows that around 25% of DNS lookups take more than 1 s and as many as 5% of DNS lookups take more than 10 s. DNS pre-fetching removes this excessive variability, represented by the height of the bubbles in the diagrams. Jung *et al.* [17] found that around 85% of lookups were resolved locally. We have no data for the benefit of DNS pre-fetching in a LAN setting. This benefit is likely to be small because the resolution requests, if done at all, will probably be local as well. DNS lookups create initial latency for both short and longer lasting flows, but they do not affect per-message latency once the initial lookup is done.

CDN (Content Distribution Network; Section II-C1):

By placing content closer to the user, CDNs can greatly reduce the latency. E.g., practical tests show that, when combined with front-end optimization (FEO), CDNs can make web pages load up to four times faster [325]. A CDN is most cost-effective for small objects, but it still reduces completion time considerably for larger objects, although with more storage capacity expense. More sophisticated solutions are available, e.g., those that serve the start of a large object from cache, then resort to the origin server for the bulk of the data. CDNs rely on economy of scale by sharing the cost over many users to increase the cache-hit-rate. Therefore, a CDN would be prohibitively expensive and rarely improve latency for content that is already local, which is why a CDN would provide no benefit for objects retrieved over a LAN.

Data pre-fetch (Section II-C2):

Pre-fetching data can offer near-zero latency, but only if the right data was included in all the data that was pre-fetched. Arbitrarily large reductions in delay can be achieved by pre-fetching arbitrarily large amounts of data, which is why the bubbles for pre-fetching data show high gain but also high pain—towards the costly left-hand end of the plots. The precise levels of gain and cost will be highly scenario-specific (e.g., more interesting over DSL than cellular access). The one certainty is that pre-fetching more data offers diminishing returns.

2) *Interactions Between Endpoints:*

TFO (TCP Fast Open; Section III-A3):

The rightmost column of Table I shows that TFO removes the initial round trip, but only for a resumed session. A 20 kB transfer will be broken into 14 segments if the maximum transmission unit is 1500 B. For a TCP slow start with a traditional initial window of 3 segments, it will take 3 rounds of window doubling to transfer these segments (3, 6 & 5 segments respectively). So in this case TFO reduces completion time by roughly 1–3/4 or 25%. Of course, for connections to new servers or to servers that have refreshed the seed for the TFO-cookie, there will be no reduction, hence the bubble in the diagram extends down to zero. On a LAN, the best latency reduction will be a little less than 25% merely because server processing time becomes more significant relative to round trips. For any large transfer, the single round-trip saved by TFO makes little difference.

TLS-FS (Transport Layer Security/False-Start; Section III-B1):

As can be seen in Table I, False-Start saves a round trip when first opening a secure session. However, when resuming a session, the existing TLS design already saves a round trip and False-Start does not improve on this (except in the less common case where the server resumes the session). Therefore, the gain of False-Start is similar to that of TFO, but in the converse circumstances (only for new, not resumed sessions). Like TFO, False-Start only saves delay if both ends support it, but it suffers from considerable additional deployment constraints, as explained in Section III-B1.

RTOR (Retransmission TimeOut Restart) and TLP (Tail Loss Probe) (Section III-C2):

Techniques related to enhanced packet loss recovery only come into play when loss occurs. As a result their average gain is low, but they can bring significant gains for the short flows hit by loss(es), thus making low delay much more predictable.

In the typical case RTOR shaves one RTT off the recovery time when a loss occurs in the end of a flow, and up to one RTT plus the delayed ACK time in the best case [97]. For a short flow over a WAN, assuming an RTO of twice the RTT and a delayed ACK time of 200 ms, this will result in a 25% reduction in completion time in the best case. Over a LAN, loss is more rare and the RTT is less significant in relation to the RTO. The delayed ACK time can, however, be significant. Assuming a Linux sender with a minimum RTO of 200 ms, the flow completion time of a short flow can in the best case be almost halved by removing the delayed ACK time from the recovery time.

As reported in Flach *et al.* [91], TLP (Tail Loss Probe) reduces the average completion time for short Web flows over the Internet by roughly 3%. Furthermore, [91] reports observing RTOs that are 200 times the RTT in their Web traces. As TLP can reduce the loss recovery to a few RTTs the gain in the best case can be close to 100%. Over a LAN, the gain in the best case can also be close to 100%, as the RTO is typically much larger than the RTT.

For large transfers the loss recovery delay is of limited impact, rendering both schemes less significant for this case. Both schemes are straightforward to deploy as they require only sender-side modifications.

Adding Forward Error Correction (FEC) to TCP (Section III-C3):

Mechanisms for adding FEC to TCP can fully mask packet loss when it occurs. As the packet loss recovery time can dominate the completion time for short flows in both WAN and LAN environments, the gain is in the best case close to 100%. The average gain and the impact on long transfers is still limited for reasons discussed above. Adding FEC to TCP requires support at both the sender and receiver. Care must also be taken to ensure that proper congestion control is invoked for packets that are lost and recovered through FEC.

ECN (Explicit Congestion Notification; Section III-C4):

The use of ECN can also remove the need for dropping packets. As currently specified [63], ECN will have similar gains to TCP with FEC, in that it removes the packet loss recovery time. The use of ECN requires support from both ends as well as the network.

3) *Transmission Path:*

Straighter links (Section IV-A1):

A common rule of thumb used within carriers is that a fiber or cable route will be 25% longer than the line-of-sight route, but no scientific references to this are known. The shorter transmission path would cut the completion time of short TCP flows by $(1 - 1/1.25 = 20\%)$, because a short flow is still in the slow-start phase when it finishes, which requires a set number of round trips for a certain transfer size. The completion time of a larger transfer is limited by bandwidth, not path length, so straighter cables would typically not affect its completion time. In the case where two long-running TCP Reno transfers share a bottleneck they will take shares of capacity inversely proportional to RTT, so the completion time of a large transfer would be shorter if the cables were straighter. However, the rate of modern TCP is becoming less dependent on RTT (e.g., TCP CUBIC's packet rate is roughly proportional to $1/\mathrm{RTT}^{0.25}$ [326]).

Hollow fiber (Section IV-A2):

The signal velocity in ordinary optical fiber is 2/3 of that in air or hollow fiber. Therefore, hollow fiber could reduce path delay by up to 33%, assuming its jointing losses were improved. This advantage would typically only be realized by short flows, for the same reasons as discussed above for straighter links. Both hollow fiber and straighter links would be prohibitively expensive, except for niche applications such as private links between financial centers.

Microwave (Section IV-A3):

Microwave combines the 20% line-of-sight benefit (Section IV-A1) and the 33% increase in signal velocity, making a total reduction in delay of roughly $(100\% - 20\%)(100\% - 33\%) = 53\%$. This has to be traded off against weather-induced unreliability. Therefore, in practice, the theoretical 53% latency gain from microwave is often reduced by the need for repeaters (which add 6.5 μs every 60 km) and the need for end-to-end link FEC (adding around 150 μs).

Reducing WiFi medium acquisition delay (Section IV-B):

The IEEE 802.11 Medium Access Control incorporates two medium access methods: the mandatory Distributed Coordination Function (DCF) method and the optional Point Coordination Function (PCF) which provides Time Bounded Services (TBS). DCF is based on the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol, with two possible access methods, a so-called basic method and the RTS/CTS method. Vardakas *et al.* [327] shows that, depending on the data rate and the number of stations, delays can be reduced by hundreds of milliseconds up to more than a second using RTS/CTS instead of the basic method.

Adopted in IEEE 802.11n, frame aggregation improves efficiency by reducing overhead due to MAC headers. Gains due to frame aggregation depend strongly on the rate and noise level of the channels, number of stations, packet size and specifics of aggregation technique [118]. [328] shows that dozens to hundreds of milliseconds can be removed using two-level aggregation instead of A-MPDU, while [117], [329] show that two-level aggregation outperforms A-MPDU, which outperforms A-MSDU aggregation. Moreover, [330] experimentally compares a system with and without frame aggregation, and shows that frame aggregation improves throughput, with minimum impact on average inter-frame delay, albeit with some impact on jitter. Other works [119], [331] show that frame aggregation can be deployed and fulfill delay sensitive requirements, such as for voice and video services.

Removing DSL Interleaving (Section IV-D):

Forward error correction coupled with interleaving (I-FEC) is a mechanism widely adopted for impulse noise protection. On the one hand, the mechanism itself introduces delay but, on the other hand, it hides transmission errors along wired or wireless media, which would otherwise introduce retransmission and time-out delays. Interleaving introduces the following levels of one way delay:

- ADSL1: operators use either FAST (0 ms interleaving) or SLOW (16 ms interleaving).
- ADSL2/2+: most common is 8 ms interleaving.
- VDSL2: most common is 8 ms interleaving.

ARQ: retransmission is used for impulse noise protection, a shorter I-FEC with 0.5 ms interleaving delay kept for channel protection. In case a packet is retransmitted it will take 4 ms longer (so a total delay of 4.5 ms), but such retransmits are only expected to occur for 1% to 5% of the packets.

Removing interleaving on DSL is applicable only for the WAN cases. In the first case (Fig. 21(a)) removing interleaving would represent a considerable reduction in delay (e.g., removing 8 ms one-way interleaving delay removes about 8% of a 200 ms WAN round-trip time, and as much as 80% of a 20 ms round-trip time). Figure 12 of Sundaresan *et al.* [332] shows that US broadband service providers using cable can offer less than 10 ms last-mile round-trip latency to nearly 100% of their users, while a significant proportion of customers of DSL-based providers experience up to 60 ms, largely because of interleaving.

In the case of elephant file transfers (Fig. 21(c)), the reduced RTT should make little difference to throughput and therefore completion time. While removing interleaving leaves the path more susceptible to impulse noise, this would have little effect on the long-running throughput of modern more robust transport protocols, such as TCP CUBIC, which would strive to maintain the line rate, as long as the bit error rate (BER) introduces loss at a lower order of magnitude than the congestion control mechanisms themselves induce to fill the link.

Fairhurst and Wood [92] discusses the impact of a range of Automatic Repeat Request (ARQ) methods on TCP, and identifies the ways in which ARQ introduces delay to Internet traffic.

Optimize Forwarding (Sections IV-E and IV-C):

The objective is to reduce $S_{\text{total}} = S_D + S_I + S_L + S_F + S_O + S_S$, where S_L is the delay due to the packet header manipulation and S_F is the delay of passing through the switching fabric (see Section IV-E). Ramaswamy *et al.* [132] give a good overview of baseline network processing delay, while Rumble *et al.* [305] set the challenge on how far we should aim. This is closely related to improvements addressed in Section VI-C on OS delays. These delays are of an order of magnitude of nanoseconds, thus improvements in these sources will be small relative to typical delays from other sources measured in microseconds. S_I and S_O represent the input and output port queuing delays, in addition serialization/deserialization delays are given by S_S and S_D. Newman [333] indicates that, if cut-though is properly implemented, it can remove dozens to hundreds of microseconds on a 10 GE interface. Moreover, I/O buffering and serialization/deserialization delays can be reduced by increasing the port line speed or reducing the number of port traversals (see Section IV-C). In comparison with other sources of delays, which can reduce latency in the milliseconds order of magnitude, improvements in forwarding yield small delay gains.

Correct buffer sizing (Sections IV-F2 and IV-F3):

The work of [135] and others has shown that buffers in many places in the Internet are sized far larger than they should be. Unless the traffic pattern makes one of these buffers become the bottleneck, they will introduce no delay. However, whenever traffic converges on such a buffer, long delays will be experienced—as long as a few seconds. No buffer ever needs to hold more data than the longest possible round-trip half-way round the earth and back at the speed of light in glass (about 200 ms). Therefore, re-sizing a 2 s buffer to 200 ms will immediately reduce worst-case latency for short flows with a 200 ms base delay (case 1a) by $1 - (200 + 200)/(2000 + 200) = 82\%$.

Ideally, AQM would also be implemented in the buffer (see Section IV-F6). Adaptively sizing low level buffers to the minimum needed to absorb queue variation can both significantly reduce delay and push the queue up into a higher layer buffer where AQM is more appropriate (see Section IV-F2). Correct sizing of a buffer should always be possible by configuration, even if AQM or adaptive buffer sizing has not been implemented in the equipment at hand. A buffer intended for numerous multiplexed flows may be sized even smaller than one worst-case global round-trip (see Section IV-F3).

Class-based scheduling (Section IV-F4):

Prioritizing latency-sensitive traffic can eliminate most queuing delays along a path, provided that the latency sensitive traffic uses less than the capacity available. The latency reduction comes at the expense of extra latency for other traffic sharing the path. This can eliminate close to 100% of queuing delay for latency-sensitive traffic along the path. The percentage reduction of the overall flight latency depends on what the minimum flight time is when there is no queuing, so it can provide savings of between $\sim$95% and $\sim$50% with current levels of network buffering.

Class-based scheduling is only applicable where a small proportion of traffic is latency sensitive and the remainder is not. In scenarios where the majority of the traffic is latency sensitive, it is less useful.

AQM (Active Queue Management; Section IV-F6):

AQM algorithms aim at avoiding having buffers that are persistently full, and recent AQM proposals use queuing delay as an input variable to their control system [189], [190]. For instance, PIE—like Adaptive RED, a much older algorithm [334]—tries to stabilize the queue around a preset value of a few tens of ms. Hence, any reduction in queuing delay achieved thanks to AQM is to be compared to the maximum delay corresponding to a full buffer, and is dependent on the way the AQM parameters are set; other parameters such as the load and degree of statistical multiplexing (i.e., number of flows and composition of the traffic mix) will also play a role in how (much) the buffer is filled— e.g., an AQM may have little effect in a very lightly-loaded buffer.

In the absence of bufferbloat, buffers for small numbers of flows can be supposed to have been sized as to absorb (roughly) one worst-case RTT's worth of packets, which gives an indication of the maximum possible queue length. In the WAN example, if we consider a persistently filled 200 ms buffer (i.e., an RTT-sized buffer) and an AQM achieving an average queue of 20 ms, then the four rounds needed to transfer 20 kB of data with TCP (one for connection setup, plus three for the data) will take about 1600 and 880 ms without and with AQM, respectively. This amounts to $\sim$45% reduction in completion time. With a bloated bottleneck buffer—say, one second's worth of buffer space—completion time without AQM may take as much as $\sim$4.8 s, hence giving a reduction of $\sim$83% when AQM is used. Similar rough estimates can be obtained in the LAN scenario with short flows, 2 ms buffers and an AQM that keeps queuing delays at a few hundreds of μs (like DCTCP's does at 1 Gbps link speeds [194]).

DCTCP (Data Centre TCP; Section IV-F6):

The developers of DCTCP claim that it cut mean completion time in their Bing data centre by about 40%–45% for short flows and by about 86% during periods when load was ten times the normal load. Perhaps more importantly, the 99.9th percentile delay reduced by over 40%, meaning that delay not only reduced, but it became more predictably low. DCTCP can be deployed in a private data centre, but its placement on the bubble diagram indicates that it would be very hard to deploy in a multi-tenant data centre or on the public Internet. This is because DCTCP requires simultaneous changes to senders, receivers and all switches. A way has been proposed to incrementally deploy DCTCP on the public Internet [209], which would shift the DCTCP bubble to the right. However, it is only in the early stages of evaluation at the time of writing.

DBCC (Delay-Based Congestion Control; Section IV-F7c):

Delay-based congestion controls use path delay (one-way or round-trip) as a measure of congestion. During periods when all flows at a queue are using DBCC, this enables them to keep queuing delays along the path low; often to below a configurable threshold of a few milliseconds. For example, if the bottleneck link has a maximum queuing delay of 200 ms, and the sources use delay-based congestion control with a threshold of 10 ms, there will be a 95% reduction in maximum queuing delay compared to what would happen if loss-based congestion control was used. The overall flight latency reduction depends on the minimum flight time when there is no queuing. If this was 2 ms, for example, the saving could be $\sim$95%, and for a semi-global connection of, e.g., 200 ms, a ∼47% reduction could be achieved. However, DBCC can have performance issues when coexisting with conventional loss-based approaches (see Section IV-F7c).

4) Related to Link Capacities:

IW10 (TCP initial window = 10; Section V-C2):

A short 20 kB flow typically consists of 14 segments. Therefore, following TCP's initial handshaking round, it would require 2 rounds to complete (consisting of 10 then 4 segments respectively). This compares to the 3 rounds necessary if IW were 3 (see earlier). Therefore in this case, IW10 cuts delay by 25%. The large majority of web flows are ten segments or less [246], therefore they would complete in one round (not including the handshake round). The potential negative effects of using a larger initial window are not yet fully understood, requiring further evaluations [247].

IW10+TFO:

For a resumed 20 kB connection, combining IW10 with TFO would cut both the handshake round and one round of slow-start relative to IW3, which leads to a significant halving of completion time in the case of a 20 kB flow.

IW10+TLS-FS+TFO:

Combining all three of these techniques would cut two rounds from both a first-time connection and a resumed connection.

QS (Quick-Start; Section V-C2):

As simulated and shown analytically in Sarolahti *et al.* [263], Quick-Start will satisfy a request for a small 20 kB flow in two round trips, which is 50% of the four round trips that regular TCP slow-start would take. Quick-Start cannot complete in less than two round-trips, because it needs the first round trip for the explicit signals to request and grant the sending rate. Over a LAN, the full 50% gain may not be realized if the bottleneck rate is low. For instance, after the initial rate has been granted, a 20 kB flow will take 1.5 ms to transfer at 100 Mb/s, which consumes nearly another round trip, leading to only 35% reduction in latency. Quick-Start offers little benefit to elephant file transfers, because the start-up latency becomes a small proportion of the overall completion time.

Despite the considerable gain of Quick-Start, the deployment pain is high, except for private networks. A host cannot know whether it is safe to send at the granted rate [335], if it may overrun a L2 bottleneck that has not been updated to support Quick-Start. The Quick-Start protocol maintains a count of QS hops so that a host will not use the granted rate if the number of QS hops is less than the number of IP hops (taken from the IP time-to-live field). During initial deployment when very few paths will be fully populated with QS-enabled routers, this will nearly always prevent QS from being used across a general Internet path. Layer 3 approaches like Quick Start (Section V-C2) do not check whether lower layer buffers can accept a sudden influx of traffic.

New-CWV (New Congestion Window Validation; Section V-D2):

Using new-CWV benefits other flows sharing a bottleneck by reducing collateral damage, but can also reduce the latency of a flow itself: for example, a flow that downloads a series of Web objects, with a request for 20 kB every 10 s would experience a delay of 4 round trips per request using either standard TCP or the experimental congestion window validation update in RFC2861. A persistent connection could complete each transfer in 1 RTT using new-CWV. The benefit of using new-CWV is strongly dependent on the traffic pattern and network conditions. More scenarios and results are presented in Biswas *et al.* [336] and Angelogiannopoulos [337] who reported a 60% improvement for YouTube traffic.

5) Intra-End-Host:

Stack buffering (Section VI-A):

Appropriate dimensioning and handling of end-host buffering can bring significant latency improvements for some particular applications. For instance, zero-copying techniques in the context of web servers (i.e., avoiding memory copying and context switching between processes that constitute a server-side application) have been shown in [290] to allow for speedup factors between 1.3 and 2.3, in terms of number of served requests; similarly, they show that a zero-copying technique allows to serve up to 44% more requests with a "good" response time (as specified in a standard web benchmark) under similar conditions of load, file sizes, etc. Moreover, [338] recently demonstrated that a redesign of how the TCP stack is implemented in the OS (kernel and user-space), taking into account multicore processors, and addressing inefficiencies from packet I/O, memory and TCP connection management, can dramatically improve host performance by 33% to 320% compared to the plain Linux stack. However, given the information available in the above cited references, it is very difficult to infer how such performance gains would translate into end-to-end latency gains for a specific flow.

Multi-streaming (Section VI-B):

Application layer framing (ALF) and the potential to deliver data out-of-order in the Stream Control Transport Protocol (SCTP [94]), SPDY (Section VII-A) or QUIC (Section VII-B) can allow multiple ordered data streams to be efficiently multiplexed onto a single transport association or connection; this means that these streams are handled by the same congestion controller, as opposed to each one getting their own. A flow could only see a benefit if it consists of several smaller streams; in this case, the completion time heavily depends on packet loss [307] as well as how the session is divided (how many streams, of which length, and when they begin).

To better understand the range of possible benefits, we can consider the worst and best possible case in terms of starting times: if the streams begin at exactly the same time, there will typically be no benefit from using multi-streaming as opposed to using multiple separate transport associations or connections—in fact, the slightly less aggressive congestion control behavior of one as opposed to two controllers could lead to a slightly *larger* delay. If, however, the two streams are exactly consecutive, i.e., stream 2 begins when stream 1 ends, then stream 2 can benefit from immediately using the larger congestion window of stream 1.

Assume, for example, that a short 20 kB (typically 14 segments) flow is split in half and the two streams are sent sequentially. Without multi-streaming, stream 1 needs 2 rounds in Slow Start and stream 2 needs 2 rounds, yielding a total of 4 rounds. With multi-streaming, the second stream can be transferred in only 1 round, yielding a total of 1 rounds—a reduction by 25%. With larger flows, the doubling of the congestion window in Slow Start means that, ideally, flows are split in half such that half of the transfer can finish within only one round following a larger number of preceding rounds—a completion time reduction that approximates but never reaches 50%. The gain increases with the number of rounds, i.e., the length of the transfer, but so does the chance of the TCP sender leaving Slow Start due to loss or reaching the capacity limit. Remember, though, that this is only the benefit of using multi-streaming as opposed to sending the multiple streams sequentially without such mechanism.

X. Conclusion

Historically, the Internet community has worked to improve throughput and resource utilization. There is, however, a growing awareness that latency is today often the key limiting factor for user experience. This is in part driven by an increasing number of latency-sensitive interactive Web and cloud based applications, and in part by increasing amounts of capacity becoming available over the Internet. In contrast to bandwidth, where the bottleneck link determines the capacity available for a communication session, the latency experienced by a communication session is additive in nature where a number of different sources may contribute to the experienced latency. In this work we aimed to identify the different sources of delay that may affect a communication session, providing a structured overview of the latency problem, and to survey available techniques for reducing latency as well as their merits.

Structural delays, such as placement of servers and suboptimal routes can contribute significantly to latency. Structural delays are particularly problematic in less developed parts of the Internet where CDNs and other caching infrastructure are lacking and peering agreements often add significant routing stretch.

Various interactions between the endpoints are also a source of latency. For short flows, initialization delays can contribute a significant component of the experienced latency, but they do not affect per-message latency once the initialization phase is completed. While loss may only occasionally occur, packet loss recovery delays often contribute significantly to the higher percentile delays experienced by short flows and individual messages, giving significant impact on user-perceived performance. As previously discussed in Section III-C, it has been found that Web flows experiencing loss see a fivefold increase in completion times on average, making loss recovery delays a dominating factor for Web latency.

Several sources of delay accumulate along the transmission path. Here, queuing delay is of particular importance as excessive queue build-ups may increase the latency by several orders of magnitude. This has spurred renewed interest in AQM, as underlined in 2013 by the creation of an AQM Working Group in the IETF and the requirement to use AQM in the latest DOCSIS standard. Further work on AQM, as well as the interaction with transport-based queue control, are important topics moving forward.

The way in which link capacity is being used and shared also contributes to overall latency. As link capacities keep increasing while the majority of flows over the Internet remain small, scalable methods for rapidly sensing available capacity become increasingly important. Closely related to the issue of queuing delays, methods for capacity sharing that not only provide a suitable service for the flow itself, but also limit the collateral damage imposed on other flows can also impact the latency experienced.

Intra-end-host delays, in the form of protocol stack buffering and operating system delays, can contribute significant latency. Good progress is being made in removing such delays due to the efforts of the bufferbloat project among others.

The gains of select key techniques for reducing latency in relation to the difficulty or cost of deployment have been estimated in this survey. Such estimates are inherently imprecise as many factors influence the gain obtained in a specific scenario and fully comparable baseline cases are hard to establish. Nevertheless, we find the estimation important for furthering the understanding of the possible solution space and its limitations. As can be seen in Fig. 21(a) and (b), the vertical extent of the bubbles is large for many of the solutions. This is a reflection of the fact that many important sources of latency are intermittent in nature, including queuing delay and loss recovery delay. The associated solution techniques remove or reduce the resulting occasional large delays and make the latency more predictable.

Reducing latency for the benefit of Internet users constitutes a key challenge for the networking community over the coming years. As should have become clear, removing all the sources of latency is a multifaceted undertaking and will require combining the various different competencies in the scientific and industrial communities in a collective effort. By identifying key sources of latency, highlighting the various techniques that can be applied to reduce latency and outlining the gains that can be expected from such techniques in relation to the cost and difficulty of deployment, it is our hope that this paper can serve as a starting point to further drive this important work forward.

Acknowledgment

This work was inspired by an unpublished survey by Nick Gates of BT and the University of Cambridge and Bob Briscoe, Philip Eardley and Carla Di Cairano Gilfedder of BT. The work also benefited from the discussions at the ISOC Latency Workshop in London, where an early (2pp) version of the survey was presented [339], and suggestions from the anonymous reviewers and Dave Täht's public review of a preprint.

We would also like to thank James Sterbenz and Joe Touch for their excellent text on this subject, which is recommended reading; "High-speed networking—A systematic approach to high-bandwidth low-latency communication" [340].

REFERENCES

[1] E. Schurman and J. Brutlag, "Performance related changes and their user impact," in *Proc. Velocity Web Perform. Oper. Conf.*, Jun. 2009, pp. 1–13. [Online]. Available: http://tinyurl.com/yg3xbhk

[2] M. Mayer, "In Search of … a Better, Faster, Stronger Web," in *Proc. Velocity*, 2009 (Online video no longer available), Jun. 2009.

[3] T. Zou *et al.*, "Making time-stepped applications tick in the cloud," in *Proc. ACM SOCC*, 2011, pp. 1–14.

[4] E. Rosen, A. Viswanathan, and R. Callon, *Multiprotocol Label Switching Architecture*, RFC 3031 (Proposed Standard), Updated by RFCs 6178, 6790, Internet Engineering Task Force, Jan. 2001.

[5] Y. Rekhter, T. Li, and S. Hares, *A Border Gateway Protocol 4 (BGP-4)*, RFC 4271 (Draft Standard), Updated by RFCs 6286, 6608, 6793, Internet Engineering Task Force, Jan. 2006.

[6] J. Moy, *OSPF Version 2*, RFC 2328 (INTERNET STANDARD), Updated by RFCs 5709, 6549, 6845, 6860, Internet Engineering Task Force, Apr. 1998.

[7] H. Xie, L. Qiu, Y. Yang, and Y. Zhang, "On self adaptive routing in dynamic environments—An evaluation and design using a simple, probabilistic scheme," in *Proc. IEEE ICNP*, Oct. 2004, pp. 12–23.

[8] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, RFC 2992 (Informational), Internet Engineering Task Force, Nov. 2000.

[9] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault, *Multi-Topology (MT) Routing in OSPF*, RFC 4915 (Proposed Standard), Internet Engineering Task Force, Jun. 2007.

[10] F. Valera, I. V. Beijnum, A. Garcia-Martinez, and M. Bagnulo, "Multipath BGP: Motivations and solutions," in *OpenAIRE: Open Access Infrastructure for Research in Europe*, vol. 216372. Cambridge, U.K.: Cambridge Univ. Press, 2011, pp. 1–20. [Online]. Available: http://orff.uc3m.es/handle/10016/10324

[11] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *TCP Extensions for Multipath Operation With Multiple Addresses*, RFC 6824 (Experimental), Internet Engineering Task Force, Jan. 2013.

[12] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, *Architectural Guidelines for Multipath TCP Development*, RFC 6182 (Informational), Internet Engineering Task Force, Mar. 2011.

[13] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *ACM SIGCOMM CCR*, vol. 35, no. 2, pp. 5–12, Apr. 2005. [Online]. Available: http://www.statslab.cam.ac.uk/~frank/PAPERS/kv.html

[14] Apple, *iOS: Multipath TCP support in iOS 7*, Cupertino, CA, USA, Jan. 2014. [Online]. Available: http://support.apple.com/kb/HT5977

[15] M. Scharf and A. Ford, *Multipath TCP (MPTCP) Application Interface Considerations*, RFC 6897 (Informational), Internet Engineering Task Force, Mar. 2013.

[16] M. X. Makkes, A. Oprescu, R. Srijkers, and R. Meijer, "MeTRO: Low latency network paths with routers-on-demand," in *Proc. Eur. –Par , Parallel Process. Workshops*, *Springer Lecture Notes in Computer Science*, Oct. 2013, pp. 333–342.

[17] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 589–603, Oct. 2002.

[18] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, "Comparing DNS resolvers in the wild," in *Proc. ACM IMC*, Melbourne, Vic., Australia, 2010, pp. 15–21.

[19] T. Callahan, M. Allman, and M. Rabinovich, "On modern DNS behavior and properties," *ACM SIGCOMM CCR*, vol. 43, no. 3, pp. 7–15, Jul. 2013.

[20] S. Sundaresan, N. Magharei, N. Feamster, and R. Teixeira, "Accelerating last-mile web performance with popularity-based prefetching," *ACM SIGCOMM CCR*, vol. 42, no. 4, pp. 303–304, Aug. 2012.

[21] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010.

[22] J. S. Otto, M. A. Sánchez, J. P. Rula, and F. E. Bustamante, "Content delivery and the natural evolution of DNS: Remote DNS trends, performance issues and alternative solutions," in *Proc. ACM IMC*, Boston, MA, USA, 2012, pp. 523–536.

[23] G. Barish and K. Obraczka, "World wide web caching: Trends and techniques," *IEEE Commun. Mag.*, vol. 38, no. 5, pp. 178–184, May 2000.

[24] S. Podlipnig and L. Böszörmenyi, "Replacement strategies for quality based video caching," in *Proc. IEEE ICME*, 2002, vol. 2, pp. 49–52.

[25] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM CSUR*, vol. 35, no. 4, pp. 374–398, Dec. 2003.

[26] T. M. Kroeger, D. D. Long, and J. C. Mogul, "Exploring the bounds of web latency reduction from caching and prefetching," in *Proc. USITS*, 1997, pp. 13–22.

[27] A. Dan, M. G. Kienzle, and D. Sitaram, "A dynamic policy of segment replication for load-balancing in video-on-demand servers," *Multimedia Syst.*, vol. 3, no. 3, pp. 93–103, Jul. 1995.

[28] L. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The Google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, Mar. 2003.

[29] L. Kontothanassis, *Content Delivery Considerations for Different Types of Internet Video*, ACM MMSys, Portland, OR, USA, 2012. [Online]. Available: http://www.mmsys.org/?q=node/64

[30] Y. Chen, S. Jain, V. K. Adhikari, and Z.-L. Zhang, "Characterizing roles of front-end servers in end-to-end performance of dynamic content distribution," in *Proc. ACM IMC*, 2011, pp. 559–568.

[31] M. Saxena, U. Sharan, and S. Fahmy, "Analyzing video services in Web 2.0: A global perspective," in *Proc. ACM NOSSDAV*, 2008, pp. 39–44.

[32] P. B. Beskow, K.-H. Vik, P. Halvorsen, and C. Griwodz, "The partial migration of game state and dynamic server selection to reduce latency," *Multimedia Tools Appl.*, vol. 45, no. 1–3, pp. 83–107, Oct. 2009.

[33] Y. Chen, S. Byna, and X.-H. Sun, "Data access history cache and associated data prefetching mechanisms," in *Proc. ACM/IEEE SC*, Reno, NV, USA, 2007, pp. 1–12.

[34] F. T. Johnsen, T. Hafsøe, C. Griwodz, and P. Halvorsen, "Workload characterization for news-on-demand streaming services," in *Proc. IEEE IPCCC*, Apr. 2007, pp. 314–323.

[35] L. Pantel and L. Wolf, "On the suitability of dead reckoning schemes for games," in *Proc. Annu. Workshop NetGames*, Apr. 2002, pp. 79–84.

[36] W. Palant, C. Griwodz, and P. Halvorsen, "Evaluating dead reckoning variations with a multi-player game simulator," in *Proc. ACM NOSSDAV*, B. N. Levine and M. Claypool, Eds., 2006, pp. 20–25.

[37] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 600–608, Aug. 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1015706.1015766

[38] D. Cohen-Or, "Model-based view-extrapolation for interactive VR Web-systems," in *Proc. Comput. Graph. Int.*, 1997, pp. 104–112. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=601282

[39] D. Cohen-Or, *Model-based view extrapolation for interactive virtual reality systems*, U.S. Patent 6 307 567 B1, Oct. 23, 2001. [Online]. Available: http://www.google.com/patents/US6307567

[40] A. Kumar, S. Merugu, J. J. Xu, E. W. Zegura, and X. Yu, "Ulysses: A robust, low-diameter, low-latency peer-to-peer network," *Eur. Trans. Telecommun.*, vol. 15, no. 6, pp. 571–587, Nov./Dec. 2004.

[41] T. Small, B. Li, and B. Liang, "Outreach: Peer-to peer topology construction towards minimized server bandwidth costs," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 1, pp. 35–45, Jan. 2007.

[42] K.-H. Vik, "Group communication techniques in overlay networks," Ph.D. dissertation, Simula Research Laboratory/University of Oslo, Oslo, Norway, Dec. 2008. [Online]. Available: https://simula.no/publications/Simula.simula.20

[43] *Application-Layer Traffic Optimization (alto)*, 2014. [Online]. Available: http://datatracker.ietf.org/wg/alto/

[44] Amazon, *Amazon EC2 instance types*, Seattle, WA, USA. [Online]. Available: http://aws.amazon.com/ec2/instance-types/

[45] Microsoft, *Windows azure*, Redmond, WA, USA, 2013. [Online]. Available: http://www.windowsazure.com/

[46] Google, *Google compute engine*, Mountain View, CA, USA, 2013. [Online]. Available: https://cloud.google.com/products/compute-engine

[47] G. Armitage, "Optimising online FPS game server discovery through clustering servers by origin autonomous system," in *Proc. ACM NOSSDAV*, Braunschweig, Germany, May 2008, pp. 3–8.

[48] P. Beskow, A. Petlund, G. Erikstad, C. Griwodz, and P. Halvorsen, "Reducing game latency by migration, core-selection and TCP modifications," *IJAMC*, vol. 4, no. 4, pp. 343–363, Nov. 2010.

[49] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.

[50] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proc. ACM MMSys*, Phoenix, AZ, USA, 2010, pp. 35–46.
[51] K. Raaen, A. Petlund, and P. Halvorsen, "Is today's public cloud suited to deploy hardcore realtime services?" in *Proc. Eur. –Par , Parallel Process. Workshops*, *Lecture Notes in Computer Science*, 2013, pp. 343–352.
[52] "Single root I/O virtualization," PCI-SIG, Specification v1.1 2009. [Online]. Available: http://www.pcisig.com/members/downloads/specifications/iov/sr-iov1_1_20Jan10.pdf
[53] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246 (Proposed Standard), Updated by RFCs 5746, 5878, 6176, Internet Engineering Task Force, Aug. 2008.
[54] W. M. Petullo, X. Zhang, J. A. Solworth, D. J. Bernstein, and T. Lange, *MinimaLT: Minimal-latency networking through better security*, Univ. Illinois Chicago, Chicago, IL, USA, Cryptology ePrint Archive, Rep. 2013/310, May 2013. [Online]. Available: http://eprint.iacr.org/2013/310
[55] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," Internet Draft draft-ietf-tcpm-fastopen, Jul. 2014, Work in progress.
[56] S. Frankel and S. Krishnan, *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap*, RFC 6071 (Informational), Internet Engineering Task Force, Feb. 2011.
[57] J. Postel, *Transmission Control Protocol*, RFC 793 (INTERNET STANDARD), Updated by RFCs 1122, 3168, 6093, 6528, Internet Engineering Task Force, Sep. 1981.
[58] A. Langley, N. Modadugu, and B. Moeller, "Transport Layer Security (TLS) False Start," Internet Draft draft-bmoeller-tls-falsestart, Jun. 2010, Work in progress.
[59] A. Langley, "Transport Layer Security (TLS) Snap Start," Internet Draft draft-agl-tls-snapstart-00, Jun. 2010, Work in progress.
[60] A. Bittau *et al.*, "Cryptographic Protection of TCP Streams (tcpcrypt)," Internet Draft draft-bittau-tcp-crypt-03, Sep. 2012, Work in progress.
[61] R. Stewart, *Stream Control Transmission Protocol*, RFC 4960 (Proposed Standard), Updated by RFCs 6096, 6335, Internet Engineering Task Force, Sep. 2007.
[62] E. Kohler, M. Handley, and S. Floyd, *Datagram Congestion Control Protocol (DCCP)*, RFC 4340 (Proposed Standard), Updated by RFCs 5595, 5596, 6335, 6773, Internet Engineering Task Force, Mar. 2006.
[63] K. Ramakrishnan, S. Floyd, and D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, RFC 3168 (Proposed Standard), Updated by RFCs 4301, 6040, Internet Engineering Task Force, Sep. 2001.
[64] D. Wing and A. Yourtchenko, *Happy Eyeballs: Success With Dual-Stack Hosts*, RFC 6555 (Proposed Standard), Internet Engineering Task Force, Apr. 2012.
[65] D. Wing and P. Natarajan, *Happy Eyeballs: Trending Towards Success With SCTP*, Internet Draft draft-wing-tsvwg-happy-eyeballs-sctp, Oct. 2010, Work in progress.
[66] J. Rosenberg *et al.*, *SIP: Session Initiation Protocol*, RFC 3261 (Proposed Standard), Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878, Internet Engineering Task Force, Jun. 2002.
[67] J. Rosenberg, *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*, RFC 5245 (Proposed Standard), Updated by RFC 6336, Internet Engineering Task Force, Apr. 2010.
[68] S. Guha and P. Francis, "Characterization and measurement of TCP traversal through NATs and firewalls," in *Proc. ACM IMC*, Berkeley, CA, USA, 2005, pp. 199–211.
[69] J. Maenpaa, V. Andersson, G. Camarillo, and A. Keranen, "Impact of network address translator traversal on delays in peer-to-peer session initiation protocol," in *Proc. IEEE GLOBECOM*, 2010, pp. 1–6.
[70] R. Mahy, P. Matthews, and J. Rosenberg, *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*, RFC 5766 (Proposed Standard), Internet Engineering Task Force, Apr. 2010.
[71] R. Braden, *T/TCP—TCP Extensions for Transactions Functional Specification*, RFC 1644 (Historic), Obsoleted by RFC 6247, Internet Engineering Task Force, Jul. 1994.
[72] C. Hannum, "T/TCP vulnerabilities," *Phrack Mag.*, vol. 8, no. 53, Jul. 1998. [Online]. Available: http://phrack.org/issues/53/6.html
[73] L. Eggert, *Moving the Undeployed TCP Extensions RFC 1072, RFC 1106, RFC 1110, RFC 1145, RFC 1146, RFC 1379, RFC 1644, RFC 1693 to Historic Status*, RFC 6247 (Informational), Internet Engineering Task Force, May 2011.
[74] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP Fast Open," in *Proc. ACM Int. CoNEXT*, Tokyo, Japan, 2011, pp. 1–12.
[75] W. Zhou, Q. Li, M. Caesar, and P. Godfrey, "ASAP: A low-latency transport layer," in *Proc. ACM SIGCOMM*, pp. 390–391.
[76] T. Berners-Lee, R. Fielding, and H. Frystyk, *Hypertext Transfer Protocol—HTTP/1.0*, RFC 1945 (Informational), Internet Engineering Task Force, May 1996.
[77] R. Fielding *et al.*, *Hypertext Transfer Protocol—HTTP/1.1*, RFC 2616 (Draft Standard), Updated by RFCs 2817, 5785, 6266, 6585, Internet Engineering Task Force, Jun. 1999.
[78] M. Belshe and R. Peon, "SPDY Protocol," Internet Draft draft-mbelshe-httpbis-spdy, Feb. 2012, Work in progress.
[79] K. Lahey, *TCP Problems With Path MTU Discovery*, RFC 2923 (Informational), Internet Engineering Task Force, Sep. 2000.
[80] P. Savola, *MTU and Fragmentation Issues With In-the-Network Tunneling*, RFC 4459 (Informational), Internet Engineering Task Force, Apr. 2006.
[81] M. Mathis and J. Heffner, *Packetization Layer Path MTU Discovery*, RFC 4821 (Proposed Standard), Internet Engineering Task Force, Mar. 2007.
[82] J. Touch and M. Townsley, "Tunnels in the Internet Architecture," Internet Draft draft-ietf-intarea-tunnels-00, Mar. 2010, Work in progress.
[83] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*, RFC 6347 (Proposed Standard), Internet Engineering Task Force, Jan. 2012.
[84] A. Langley, *TLS Next Protocol Negotiation*, Google Technical Note: nextprotoneg, Jul. 2011.
[85] A. Langley, *Transport Layer Security (TLS) next protocol negotiation extension*, Internet Draft draft-agl-tls-nextprotoneg-04, May 2012, Work in progress. [Online]. Available: http://tools.ietf.org/html/draft-agl-tls-nextprotoneg
[86] S. Friedl, A. Popov, A. Langley, and E. Stephan, "Transport Layer Security (TLS) application layer protocol negotiation extension," Internet Draft draft-ietf-tls-applayerprotoneg, Mar. 2014, (Work in progress). [Online]. Available: http://tools.ietf.org/html/draft-ietf-tls-applayerprotoneg
[87] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, *Transport Layer Security (TLS) Session Resumption Without Server-Side State*, RFC 5077 (Proposed Standard), Internet Engineering Task Force, Jan. 2008.
[88] E. Stark, L.-S. Huang, D. Israni, C. Jackson, and D. Boneh, "The case for prefetching and prevalidating TLS server certificates," in *Proc. NDSS Symp.*, 2012, pp. 1–12.
[89] E. Rescorla, "New Handshake Flows for TLS 1.3," Internet Draft draft-rescorla-tls13-new-flows, Feb. 2014, Work in progress. [Online]. Available: https://tools.ietf.org/html/draft-rescorla-tls13-new-flows
[90] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson, and G. Fairhurst, *The Lightweight User Datagram Protocol (UDP-Lite)*, RFC 3828 (Proposed Standard), Updated by RFC 6335, Internet Engineering Task Force, Jul. 2004.
[91] T. Flach *et al.*, "Reducing web latency: The virtue of gentle aggression," in *Proc. ACM SIGCOMM*, Hong Kong, 2013, pp. 159–170.
[92] G. Fairhurst and L. Wood, *Advice to Link Designers on Link Automatic Repeat reQuest (ARQ)*, RFC 3366 (Best Current Practice), Internet Engineering Task Force, Aug. 2002.
[93] M. Watson, A. Begen, and V. Roca, *Forward Error Correction (FEC) Framework*, RFC 6363 (Proposed Standard), Internet Engineering Task Force, Oct. 2011.
[94] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad, *Stream Control Transmission Protocol (SCTP) Partial Reliability Extension*, RFC 3758 (Proposed Standard), Internet Engineering Task Force, May 2004.
[95] B. Mukherjee and T. Brecht, "Time-lined TCP for the TCP-friendly delivery of streaming media," in *Proc. IEEE ICNP*, Nov. 2000, pp. 165–176.
[96] D. McCreary, K. Li, S. Watterson, and D. Lowenthal, "TCP-RC: A receiver-centered TCP protocol for delay-sensitive applications," in *Proc. SPIE/ACM Annu. MMCN*, Jan. 2005, vol. 5680, pp. 126–130.
[97] P. Hurtig, A. Brunström, A. Petlund, and M. Welzl, "TCP and SCTP RTO Restart," Internet Draft draft-ietf-tcpm-rtorestart, Jul. 2014, Work in progress. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tcpm-rtorestart
[98] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, and P. Hurtig, *Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)*, RFC 5827 (Experimental), Internet Engineering Task Force, May 2010.
[99] M. Mellia, M. Meo, and C. Casetti, "TCP smart framing: A segmentation algorithm to reduce TCP latency," *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 316–329, Apr. 2005.
[100] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis, "Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses," Internet Draft draft-dukkipati-tcpm-tcp-loss-probe, Work in

progress, Feb. 2013. [Online]. Available: http://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe

[101] O. Tickoo, V. Subramanian, S. Kalyanaraman, and K. Ramakrishnan, "LT-TCP: End-to-end framework to improve TCP performance over networks with lossy channels," in *Proc. IWQoS*, *Lecture Notes in Computer Science*, Passau, Germany, Jun. 2005, pp. 81–93.

[102] B. Ganguly, B. Holzbauer, K. Kar, and K. Battle, "Loss-Tolerant TCP (LT-TCP): Implementation and experimental evaluation," in *Proc. IEEE MILCOM*, Orlando, FL, USA, Oct. 2012, pp. 1–6.

[103] K. Evensen, A. Petlund, C. Griwodz, and P. Halvorsen, "Redundant bundling in TCP to reduce perceived latency for time-dependent thin streams," *IEEE Commun. Lett.*, vol. 12, no. 4, pp. 334–336, Apr. 2008.

[104] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM CCR*, vol. 24, no. 5, pp. 8–23, Oct. 1994.

[105] B. Briscoe, J. Kaippallimalil, and P. Thalerm, "Guidelines for Adding Congestion Notification to Protocols That Encapsulate IP," Internet Draft draft-ietf-tsvwg-ecn-encap-guidelines, Mar. 2014, Work in progress. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tsvwg-ecn-encap-guidelines-04

[106] J. Nagle, *Congestion Control in IP/TCP Internetworks*, RFC 896, Internet Engineering Task Force, Jan. 1984.

[107] R. Braden, *Requirements for Internet Hosts—Communication Layers*, RFC 1122 (Internet Standard), Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864, Internet Engineering Task Force, Oct. 1989.

[108] G. Minshall, "A Proposed Modification to Nagle's Algorithm," Internet Draft draft-minshall-nagle, Jun. 1999, Work in progress. [Online]. Available: http://tools.ietf.org/html/draft-minshall-nagle

[109] S. Cheshire, *TCP Performance Problems Caused by Interaction Between Nagle's Algorithm and Delayed ACK*, May 2005. [Online]. Available: http://www.stuartcheshire.org/papers/NagleDelayedAck/

[110] E. Ciaramella, "Wavelength conversion and all-optical regeneration: Achievements and open issues," *J. Lightw. Technol.*, vol. 30, no. 4, pp. 572–582, Feb. 2012.

[111] N. V. Wheeler *et al.*, "Wide-bandwidth, low-loss, 19-cell hollow core photonic band gap fiber and its potential for low latency data transmission," in *Proc. Nat. Fiber Opt. Eng. Conf.*, 2012, pp. 1–3.

[112] *The Importance of Dynamic Bandwidth Allocation in GPON Networks*, White Paper, PMC-Sierra, Sep. 2008.

[113] I. Rubin, *The Communications Handbook*, vol. Chapter 35, J. Gibson, Ed. New York, NY, USA: Taylor & Francis, 2002, ser. Electrical Engineering Handbook.

[114] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov, *TCP Over Second (2.5G) and Third (3 G) Generation Wireless Networks*, RFC 3481 (Best Current Practice), Internet Engineering Task Force, Feb. 2003.

[115] M. Sooriyabandara and G. Fairhurst, "Dynamics of TCP over BoD satellite networks," *Int. J. Satell. Commun. Netw.*, vol. 21, no. 4/5, pp. 427–449, Jul.–Oct. 2003.

[116] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE, Std. 802.11-2007, Jun. 12, 2007.

[117] D. Skordoulis *et al.*, "IEEE 802.11n MAC frame aggregation mechanisms for next-generation highthroughput WLANs," *IEEE Wireless Commun.*, vol. 15, no. 1, pp. 40–47, Feb. 2008.

[118] Y. Lin and V. Wong, "Frame aggregation and optimal frame size adaptation for IEEE 802.11n WLANs," in *Proc. IEEE GLOBECOM*, 2006, pp. 1–6.

[119] D. Shen *et al.*, "The performance of adaptive frame aggregation with delay limits in ultrahigh-speed WLAN," in *Proc. 12th IEEE ICCT*, 2010, pp. 1364–1368.

[120] S. Biaz and S. Wu, "Rate adaptation algorithms for IEEE 802.11 networks: A survey and comparison," in *Proc. IEEE ISCC*, Jul. 2008, pp. 130–136.

[121] E. Ancillotti, R. Bruno, and M. Conti, "Experimentation and performance evaluation of rate adaptation algorithms in wireless mesh networks," in *Proc. ACM Sym. PE-WASUN*, Vancouver, BC, Canada, 2008, pp. 7–14.

[122] E. Ancillotti, R. Bruno, and M. Conti, "Design and performance evaluation of throughput-aware rate adaptation protocols for IEEE 802.11 wireless networks," *Perform. Eval.*, vol. 66, no. 12, pp. 811–825, Dec. 2009.

[123] K. G. Shin and S. W. Daniel, "Analysis and implementation of hybrid switching," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 2, pp. 211–219, May 1995.

[124] Z. Cui, L. Xia, P. G. Bridges, P. A. Dinda, and J. R. Lange, "Optimizing overlay-based virtual networking through optimistic interrupts and cut-through forwarding," in *Proc. ACM/IEEE Conf. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Salt Lake City, UT, USA, 2012, pp. 1–11.

[125] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *Proc. 2nd IEEE/ACM Int. Symp. NOCS*, 2008, pp. 161–170.

[126] C. Shannon, "Communication in the presence of noise," *Proc. IRE*, vol. 37, no. 1, pp. 10–21, Jan. 1949.

[127] F. Foukalas, V. Gazis, and N. Alonistioti, "Crosslayer design proposals for wireless mobile networks: A survey and taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 1, pp. 70–85, 2008.

[128] D. Kliazovich, S. Redana, and F. Granelli, "Crosslayer error recovery in wireless access networks: The ARQ proxy approach," *Int. J. Commun. Syst.*, vol. 25, no. 4, pp. 461–477, Apr. 2012.

[129] "Leveraging VDSL2 for mobile backhaul: Meeting the long-term challenges in the mobile broadband era," Alcatel-Lucent, Boulogne-Billancourt, France, 2010. [Online]. Available: http://resources.alcatel-lucent.com/?cid=142941

[130] D. M. Divakaran, S. Soudan, P. Primet, and E. Altman, "A survey on core switch designs and algorithms," INRIA, Valbonne, France, Tech. Rep. RR-6942, May 2009. [Online]. Available: http://hal.inria.fr/inria-00388943

[131] N. McKeown, *Internet Routers: Past, Present and Future*, Lecture for BCS Ada Lovelace Award, Jun. 2006. [Online]. Available: http://yuba.stanford.edu/~nickm/talks/

[132] R. Ramaswamy, N. Weng, and T. Wolf, "Characterizing network processing delay," in *Proc. IEEE GLOBECOM*, Nov. 2004, vol. 3, pp. 1629–1634.

[133] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. USENIX Workshop Hot-ICE*, San Jose, CA, USA, 2012, pp. 1–6.

[134] C. Fraleigh, F. Tobagi, and C. Diot, "Provisioning IP backbone networks to support latency sensitive traffic," in *Proc. IEEE INFOCOM*, 2003, vol. 1, pp. 375–385.

[135] J. Gettys, "Bufferbloat: Dark buffers in the Internet," *IEEE Internet Comput.*, vol. 15, no. 3, pp. 96–96, May/Jun. 2011.

[136] D. Genin and J. Splett, "Where in the Internet Is Congestion?," Cornell Univ. Library, Ithaca, NY, USA, Jul. 2013. [Online]. Available: http://arxiv.org/abs/1307.3696

[137] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing residential broadband networks," in *Proc. ACM IMC*, San Diego, CA, USA, 2007, pp. 43–56.

[138] C. Kachris and I. Tomkos, "A survey on optical interconnects for data centers," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 4, pp. 1021–1036, Fourth Quarter, 2012.

[139] M. Maier and M. Reisslein, "Trends in optical switching techniques: A short survey," *IEEE Netw.*, vol. 22, no. 6, pp. 42–47, Nov. 2008.

[140] I. Chlamtac, A. Ganz, and G. Karmi, "Lightpath communications: An approach to high-bandwidth optical WANs," *IEEE Trans. Commun.*, vol. 40, no. 7, pp. 1171–1182, Jul. 1992.

[141] Y. Chen, C. Qiao, and X. Yu, "Optical burst switching: A new area in optical networking research," *IEEE Netw.*, vol. 18, no. 3, pp. 16–23, May/Jun. 2004.

[142] P. Chandra, A. Turuk, and B. Sahoo, "Survey on optical burst switching in WDM networks," in *Proc. IEEE ICIIS*, 2009, pp. 83–88.

[143] "Bufferbloat: What's wrong with the Internet?" *Commun. ACM*, vol. 55, no. 2, pp. 40–47, Feb. 2012.

[144] D. Täht, *Fixing Bufferbloat on Wireless or Not Every Packet Is Sacred*, Presentation, Nov. 2012. [Online]. Available: http://www.teklibre.com/~d/bloat/Not_every_packet_is_sacred-Battling_Bufferbloat_on_wifi.pdf

[145] *Specification for 802.3 Full Duplex Operation and Physical Layer Specification for 100 Mb/s Operation on Two Pairs of Category 3 or Better Balanced Twisted Pair Cable (100BASE-T2)*, IEEE Std. 802.3X-1997, 1997.

[146] R. Veeravalli, G. Armitage, J. But, and T. Nguyen, "Interactions Between TCP and Ethernet Flow Control Over Netgear XAVB2001 HomePlug AV links," Centre Adv. Internet Archit., Swinburne Univ. Technology, Melbourne, Vic., Australia, Tech. Rep. 130121A, Jan. 2013. [Online]. Available: http://caia.swin.edu.au/reports/130121A/CAIATR-130121A.pdf

[147] A. S. Anghel, R. Birke, D. Crisan, and M. Gusat, "Cross-layer flow and congestion control for datacenter networks," in *Proc. Workshop DC CAVES*, San Francisco, CA, USA, 2011, pp. 44–62.

[148] *Media Access Control (MAC) Bridges and Virtual Bridges*, IEEE Std. 802.1Q-2012, Dec. 13, 2012.

[149] F. Baker and G. Fairhurst, "Recommendations Regarding Active Queue Management," Internet Draft draft-ietf-aqm-recommendation, Aug. 2014, Work in progress. [Online]. Available: http://tools.ietf.org/html/draft-ietf-aqm-recommendation

[150] G. Fairhurst, R. Secchi, and A. Yun, “A flexible QoS architecture for DVB-RCS2,” *Int. J. Satell. Commun. Netw.*, vol. 31, no. 5, pp. 219–232, Sep./Oct. 2013.

[151] J. Sterbenz and G. Parulkar, “Axon: A high speed communication architecture for distributed applications,” in *Proc. IEEE INFOCOM*, San Francisco, CA, USA, 1990, pp. 415–425.

[152] D. Siemon, “Queueing in the Linux network stack,” Linux Journal, Houston, TX, USA, Sep. 2013 [Online]. Available: http://www.linuxjournal.com/content/queueing-linux-network-stack

[153] J. Corbet, “Network Transmit Queue Limits,” Aug. 2011. [Online].http://lwn.net/Articles/454390/

[154] R. Bush and D. Meyer, *Some Internet Architectural Guidelines and Philosophy*, RFC 3439 (Informational), Internet Engineering Task Force, Dec. 2002.

[155] G. Appenzeller, I. Keslassy, and N. McKeown, “Sizing router buffers,” in *Proc. ACM SIGCOMM*, Portland, OR, USA, 2004, pp. 281–292.

[156] A. Dhamdhere and C. Dovrolis, “Open issues in router buffer sizing,” *ACM SIGCOMM CCR*, vol. 36, no. 1, pp. 87–92, Jan. 2006.

[157] Y. Ganjali and N. McKeown, “Update on buffer sizing in Internet routers,” *ACM SIGCOMM CCR*, vol. 36, no. 5, pp. 67–70, Oct. 2006.

[158] A. Vishwanath, V. Sivaraman, and M. Thottan, “Perspectives on router buffer sizing: Recent results and open problems,” *ACM SIGCOMM CCR*, vol. 39, no. 2, pp. 34–39, Mar. 2009.

[159] V. Havary-Nassab, A. Koulakezian, and Y. Ganjali, “Denial of service attacks in networks with tiny buffers,” in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, 2009, pp. 91–96.

[160] K. Chandra, “Statistical multiplexing,” in *Wiley Encyclopedia of Telecommunications*. Hoboken, NJ, USA: Wiley, 2003.

[161] K. Chan, J. Babiarz, and F. Baker, *Aggregation of Diffserv Service Classes*, RFC 5127 (Informational), Internet Engineering Task Force, Feb. 2008.

[162] S. Blake *et al.*, *An Architecture for Differentiated Services*, RFC 2475 (Informational), Updated by RFC 3260, Internet Engineering Task Force, Dec. 1998.

[163] K. Nichols, S. Blake, F. Baker, and D. Black, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, RFC 2474 (Proposed Standard), Updated by RFCs 3168, 3260, Internet Engineering Task Force, Dec. 1998.

[164] R. Braden, D. Clark, and S. Shenker, *Integrated Services in the Internet Architecture: An Overview*, RFC 1633 (Informational), Internet Engineering Task Force, Jun. 1994.

[165] J. Nagle, “On packet switches with infinite storage,” *IEEE Trans. Commun.*, vol. COM-35, no. 4, pp. 435–438, Apr. 1987.

[166] A. Demers, S. Keshav, and S. Shenker, “Analysis and simulation of a fair queueing algorithm,” in *Proc. ACM SIGCOMM*, Austin, TX, USA, 1989, pp. 1–12.

[167] A. K. Parekh and R. G. Gallagher, “A generalized processor sharing approach to flow control in integrated services networks: The multiple node case,” *IEEE/ACM Trans. Netw.*, vol. 2, no. 2, pp. 137–150, Apr. 1994.

[168] P. McKenney, “Stochastic fairness queueing,” in *Proc. IEEE INFOCOM*, 1990, vol. 2, pp. 733–740.

[169] M. Kallmes, D. Towsley, and C. Cassandras, “Optimality of the last-in-first-out (LIFO) service discipline in queuing systems with real-time constraints,” in *Proc. IEEE Annu. CDC*, 1989, pp. 1073–1074.

[170] L. Huang, S. Moeller, M. J. Neely, and B. Krishnamachari, “LIFO-backpressure achieves near-optimal utility–delay tradeoff,” *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 831–844, Jun. 2013.

[171] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[172] D. Ferrari and D. Verma, “A scheme for real-time channel establishment in wide-area networks,” *IEEE J. Sel. Areas Commun.*, vol. 8, no. 3, pp. 368–379, Apr. 1990.

[173] N. Benameur, F. Guillemin, and L. Muscariello, “Latency Reduction in Home Access Gateways With Shortest Queue First,” in *Proc. ISOC Workshop Reducing Internet Latency*, Sep. 2013, pp. 1–2. [Online]. Available: http://www.Internetsociety.org/sites/default/files/pdf/accepted/4_sqf_isoc.pdf

[174] G. Carofiglio and L. Muscariello, “On the impact of TCP and per-flow scheduling on Internet performance,” *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 620–633, Apr. 2012.

[175] S. Floyd and V. Jacobson, “Link-sharing and resource management models for packet networks,” *IEEE/ACM Trans. Netw.*, vol. 3, no. 4, pp. 365–386, Aug. 1995.

[176] J. C. R. Bennett and H. Zhang, “Hierarchical packet fair queueing algorithms,” *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 675–689, Oct. 1997.

[177] I. Stoica, H. Zhang, and T. S. E. Ng, “A hierarchical fair service curve algorithm for link-sharing, real-time, priority services,” *IEEE/ACM Trans. Netw.*, vol. 8, no. 2, pp. 185–199, Apr. 2000.

[178] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements*, IEEE Std. 802.11e-2005, Nov. 11, 2005.

[179] F. Baker, J. Polk, and M. Dolly, *A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic*, RFC 5865 (Proposed Standard), Internet Engineering Task Force, May 2010.

[180] J. Turner, “New directions in communications (or which way to the information age?),” *IEEE Commun. Mag.*, vol. 24, no. 10, pp. 8–15, Oct. 1986.

[181] P. Kanuparthy and C. Dovrolis, “Shaperprobe: End to-end detection of ISP traffic shaping using active methods,” in *Proc. ACM IMC*, Berlin, Germany, 2011, pp. 473–482.

[182] M. Marcon, M. Dischinger, K. Gummadi, and A. Vahdat, “The local and global effects of traffic shaping in the Internet,” in *Proc. 3rd Int. Conf. COMSNETS*, 2011, pp. 1–10.

[183] F. Guillemin, P. Boyer, A. Dupuis, and L. Romoeuf, “Peak rate enforcement in ATM networks,” in *Proc. IEEE INFOCOM*, 1992, pp. 753–758.

[184] B. Briscoe *et al.*, “Policing congestion response in an Internetwork using refeedback,” *ACM SIGCOMM CCR*, vol. 35, no. 4, pp. 277–288, Aug. 2005.

[185] B. Briscoe, R. Woundy, A. Cooper, *Congestion Exposure (ConEx) Concepts and Use Cases*, RFC 6789 (Informational), Internet Engineering Task Force, Dec. 2012.

[186] T. V. Lakshman, A. Neidhardt, and T. Ott, “The drop from front strategy in TCP and in TCP over ATM,” in *Proc. IEEE INFOCOM*, 1996, vol. 3, pp. 1242–1250.

[187] R. Adams, “Active queue management: A survey,” *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1425–1476, 2013.

[188] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[189] R. Pan *et al.*, “PIE: A lightweight control scheme to address the bufferbloat problem,” in *Proc. IEEE 14th Int. Conf. HPSR*, Jul. 2013, pp. 148–155.

[190] K. Nichols and V. Jacobson, “Controlling queue delay,” *ACM Queue*, vol. 10, no. 5, p. 20, May 2012.

[191] D. Y. Eun and X. Wang, “Achieving 100% throughput in TCP/AQM under aggressive packet marking with small buffer,” *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 945–956, Aug. 2008.

[192] Ł. Chróst, A. Brachman, A. Chydziński. “On the performance of AQM algorithms with small buffers,” in *Springer Computer Networks*, ser. Communications in Computer and Information Science, A. Kwiecień, P. Gaj and P. Stera, Eds., vol. 39, Springer-Verlag, 2009, pp. 168–173

[193] N. Beheshti, Y. Ganjali, A. Goel, and N. McKeown, “Obtaining high throughput in networks with tiny buffers,” in *Proc. 16th IWQoS*, 2008, pp. 65–69.

[194] M. Alizadeh *et al.*, “Data Center TCP (DCTCP),” in *Proc. ACM SIGCOMM*, New Delhi, India, Sep. 2010, pp. 63–74.

[195] M. Alizadeh *et al.*, “Less is more: Trading a little bandwidth for ultra-low latency in the data center,” in *Proc. 9th USENIX Symp. NSDI*, Apr. 2012, p. 19.

[196] B. Briscoe, G. Corlianó, and B. Strulo, “How to build a virtual queue from two leaky buckets (and why one is not enough),” BT, U.K., Tech. Rep. TR-DES8-2011-001, Apr. 2012. [Online]. Available: http://www.bobbriscoe.net/projects/ipe2eqos/pcn/vq2lb/vq2lb_tr.pdf

[197] S. Islam, M. Welzl, and S. Gjessing, “One control to rule them all—Coupled congestion control for RTP media,” in *Proc. Packet Video Workshop*, San Jose, CA, USA, Dec. 2013, pp. 1–2. [Online]. Available: http://heim.ifi.uio.no/~michawe/research/publications/pv2013-fse-poster-final.pdf

[198] M. Welzl, F. Niederbacher, and S. Gjessing, “Beneficial transparent deployment of SCTP: The missing pieces,” in *Proc. IEEE GLOBECOM*, 2011, pp. 1–5.

[199] IETF, “RTP media congestion avoidance techniques (RMCAT),” IETF Working Group Charter, Sep. 2012. [Online]. Available: http://datatracker.ietf.org/doc/charter-ietf-rmcat/

[200] H. Jiang and C. Dovrolis, “Why is the Internet traffic bursty in short time scales?” in *Proc. ACM SIGMETRICS*, Banff, AB, Canada, 2005, pp. 241–252.

[201] K. Fall and S. Floyd, “Simulation-based comparisons of Tahoe, Reno and SACK TCP,” *ACM SIGCOMM CCR*, vol. 26, no. 3, pp. 5–21, Jul. 1996.

[202] K. Kobayashi, “Transmission timer approach for rate-based pacing TCP with hardware support,” in *Proc. Int. Workshop PFLDnet*, Feb. 2006, pp. 1–6. [Online]. Available: http://www.hpcc.jp/pfldnet2006/paper/s3_01.pdf

[203] A. Aggarwal, S. Savage, and T. Anderson, “Understanding the performance of TCP pacing,” in *Proc. IEEE INFOCOM*, 2000, vol. 3, pp. 1157–1165.

[204] D. Wischik, “Buffer sizing theory for bursty TCP flows,” *Int. Zurich Seminar Commun.*, pp. 98–101, 2006.

[205] D. X. Wei, P. Cao, and S. H. Low, “TCP Pacing Revisited,” 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.2658&rep=rep1&type=pdf

[206] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, “Part III: Routers with very small buffers,” *ACM SIGCOMM CCR*, vol. 35, no. 3, pp. 83–90, Jul. 2005.

[207] B. Vamanan, J. Hasan, and T. Vijaykumar, “Deadline-aware datacenter TCP (D^2TCP),” in *Proc. ACM SIGCOMM*, Aug. 2012, pp. 115–126.

[208] A. Munir *et al.*, “Minimizing flow completion times in data centers,” in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2157–2165.

[209] B. Briscoe, M. Kühlewind, D. Wagner, and J. M. R. Espinosa, “Immediate ECN,” in *Proc IETF*, Nov. 2013, pp. 1–21. [Online]. Available: http://www.ietf.org/proceedings/88/slides/slides-88-tsvwg-20.pdf

[210] C. Jin, D. Wei, and S. Low, “The case for delay-based congestion control,” in *Proc. IEEE 18th Annu. Workshop CCW*, 2003, pp. 99–104.

[211] D. Clark, M. Lambert, and L. Zhang, *NETBLT: A Bulk Data Transfer Protocol*, RFC 998 (Experimental), Internet Engineering Task Force, Mar. 1987.

[212] L. Brakmo and L. Peterson, “TCP Vegas: End to end congestion avoidance on a global Internet,” *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[213] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, *Low Extra Delay Background Transport (LEDBAT)*, RFC 6817 (Experimental), Internet Engineering Task Force, Dec. 2012.

[214] D. A. Hayes and D. Ros, “Delay-based congestion control for low latency,” in *Proc. ISOC Workshop Reducing Internet Latency*, Sep. 2013, pp. 1–2. [Online]. Available: http://www.Internetsociety.org/sites/default/files/pdf/accepted/17_delay_cc_pos-v2.pdf

[215] D. Hayes and G. Armitage, “Revisiting TCP congestion control using delay gradients,” in *Proc. IFIP Netw.*, vol. 6641, *Lecture Notes in Computer Science*, May 2011, pp. 328–341.

[216] *Link Aggregation*, IEEE Std. 802.1AX-2008, Nov. 3, 2008.

[217] M. Blanchet and P. Seite, *Multiple Interfaces and Provisioning Domains Problem Statement*, RFC 6418 (Informational), Internet Engineering Task Force, Nov. 2011.

[218] M. Wasserman and P. Seite, *Current Practices for Multiple-Interface Hosts*, RFC 6419 (Informational), Internet Engineering Task Force, Nov. 2011.

[219] C. Raiciu, M. Handley, and D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, RFC 6356 (Experimental), Internet Engineering Task Force, Oct. 2011.

[220] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, “MPTCP is not Pareto-optimal: Performance issues and a possible solution,” in *Proc. ACM 8th Int. CoNEXT*, Dec. 2012, pp. 1–12.

[221] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, “Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP,” Internet Draft draft-khalili-mptcp-congestion-control, Jul. 2014, Work in progress.

[222] J. Iyengar, P. Amer, and R. Stewart, “Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths,” *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 951–964, Oct. 2006.

[223] S. Hassayoun, J. Iyengar, and D. Ros, “Dynamic window coupling for multipath congestion control,” in *Proc. 19th IEEE ICNP*, Vancouver, BC, USA, Oct. 2011.

[224] V. Sharma, S. Kalyanaraman, K. Kar, K. Ramakrishnan, and V. Subramanian, “MPLOT: A transport protocol exploiting multipath diversity using erasure codes,” in *Proc. IEEE INFOCOM*, Phoenix, AZ, USA, Apr. 2008, pp. 592–600.

[225] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, “More is less: Reducing latency via redundancy,” in *Proc. 11th ACM Workshop HotNets*, Oct. 2012, pp. 13–18.

[226] Y.-C. Chen *et al.*, “A measurement-based study of multipath TCP performance over wireless networks,” in *Proc. ACM IMC*, Barcelona, Spain, Oct. 2013, pp. 455–468.

[227] T. Zinner, K. Tutschku, A. Nakao, and P. Tran-Gia, “Performance evaluation of packet re-ordering on concurrent multipath transmissions for transport virtualization,” *Int. J. Commun. Netw. Distrib. Syst.*, vol. 6, no. 3, pp. 322–340, Apr. 2011.

[228] F. Perotto, C. Casetti, and G. Galante, “SCTP-based transport protocols for concurrent multipath transfer,” in *Proc. IEEE WCNC*, Mar. 2007, pp. 2971–2976.

[229] A. Gurtov and T. Polishchuk, “Secure multipath transport for legacy Internet applications,” in *Proc. 6th Int. Conf. BROADNETS*, Madrid, Spain, Sep. 2009.

[230] K. Evensen, D. Kaspar, A. Hansen, C. Griwodz, and P. Halvorsen, “Using multiple links to increase the performance of bandwidth-intensive UDP-based applications,” in *Proc. IEEE ISCC*, Corfu, Greece, Jun. 2011, pp. 1117–1122.

[231] M. Coudron, S. Secci, G. Pujolle, P. Raad, and P. Gallard, “Cross-layer cooperation to boost multipath TCP performance in cloud networks,” in *Proc. IEEE Int. 2nd Conf. CloudNet*, Nov. 2013, pp. 58–66.

[232] D. Li *et al.*, “Multicast cloud with integrated multicast and unicast content distribution routing,” in *Proc. Int. Workshop Web Content Caching Distrib.*, F. Douglis and B. D. Davison, Eds., 2004, pp. 109–118.

[233] M. Hosseini, D. Ahmed, S. Shirmohammadi, and N. D. Georganas, “A survey of application-layer multicast protocols,” *IEEE Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 58–74, 2007.

[234] J. Ni and D. H. K. Tsang, “Large-scale cooperative caching and application-level multicast in multimedia content delivery networks,” *IEEE Commun. Mag.*, vol. 43, no. 5, pp. 98–105, May 2005.

[235] V. Jacobson, *Compressing TCP/IP Headers for Low-Speed Serial Links*, RFC 1144 (Proposed Standard), Internet Engineering Task Force, Feb. 1990.

[236] M. Degermark, B. Nordgren, and S. Pink, *IP Header Compression*, RFC 2507 (Proposed Standard), Internet Engineering Task Force, Feb. 1999.

[237] G. Pelletier, K. Sandlund, L.-E. Jonsson, and M. West, *RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)*, RFC 6846 (Proposed Standard), Internet Engineering Task Force, Jan. 2013.

[238] G. Pelletier and K. Sandlund, *RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite*, RFC 5225 (Proposed Standard), Internet Engineering Task Force, Apr. 2008.

[239] M. Allman, V. Paxson, and E. Blanton, *TCP Congestion Control*, RFC 5681 (Draft Standard), Internet Engineering Task Force, Sep. 2009.

[240] S. Floyd, *HighSpeed TCP for Large Congestion Windows*, RFC 3649 (Experimental), Internet Engineering Task Force, Dec. 2003.

[241] T. Kelly, “Scalable TCP: Improving performance in highspeed wide area networks,” *ACM SIGCOMM CCR*, vol. 33, no. 2, pp. 83–91, Apr. 2003.

[242] S. Ha, I. Rhee, and L. Xu, “CUBIC: A new TCP-friendly high-speed TCP variant,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[243] 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.

[244] K. Tan, J. Song, Q. Zhang, and M. Sridharan, “A compound TCP approach for high-speed and long distance networks,” in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006, pp. 1–12.

[245] S. Ha and I. Rhee, “Taming the elephants: New TCP slow start,” *Comput. Netw.*, vol. 55, no. 9, pp. 2092–2110, Jun. 2011.

[246] N. Dukkipati *et al.*, “An argument for increasing TCP’s initial congestion window,” *ACM SIGCOMM CCR*, vol. 40, no. 3, pp. 26–33, Jul. 2010.

[247] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, *Increasing TCP’s Initial Window*, RFC 6928 (Experimental), Internet Engineering Task Force, Apr. 2013.

[248] R. Sallantin *et al.*, “Safe Increase of the TCP’s Initial Window Using Initial Spreading,” Internet Draft draft-irtf-iccrg-sallantin-initial-spreading-00, Jan. 2014, Work in progress. [Online]. Available: http://tools.ietf.org/html/draft-irtf-iccrg-sallantin-initial-spreading

[249] D. Liu, M. Allman, S. Jiny, and L. Wang, “Congestion control without a startup phase,” in *Proc. Int. Workshop PFLDnet*, Feb. 2007, pp. 61–66.

[250] M. Scharf, “Comparison of end-to-end and network-supported fast startup congestion control schemes,” *Comput. Netw.*, vol. 55, no. 8, pp. 1921–1940, Jun. 2011.

[251] C. Partridge, D. Rockwell, M. Allman, R. Krishnan, and J. Sterbenz, “A swifter start for TCP,” BBN, Cambridge, MA, USA, Tech. Rep. 8339, Mar. 2002.

[252] S. Keshav, “A control-theoretic approach to flow control,” in *Proc. ACM SIGCOMM*, Zurich, Switzerland, 1991, pp. 3–15.

[253] V. Konda and J. Kaur, “RAPID: Shrinking the congestion-control timescale,” in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009, pp. 1–9.

[254] M. Kuehlewind and B. Briscoe, “Chirping for congestion control—Implementation feasibility,” in *Proc. Int. Workshop PFLDnet*, Nov. 2010, pp. 1–7.

[255] J. Touch, *TCP Control Block Interdependence*, RFC 2140 (Informational), Internet Engineering Task Force, Apr. 1997.

[256] H. Balakrishnan and S. Seshan, *The Congestion Manager*, RFC 3124 (Proposed Standard), Internet Engineering Task Force, Jun. 2001.

[257] *RTP Media Congestion Avoidance Techniques (RMCAT)*, 2014. [Online]. Available: http://datatracker.ietf.org/wg/rmcat/

[258] V. N. Padmanabhan and R. H. Katz, "TCP Fast Start: A technique for speeding up web transfers," in *Proc. IEEE Globe Internet Conf.*, 1998, pp. 1–20.

[259] I. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: A new flow control scheme for satellite networks," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 307–321, Jun. 2001.

[260] R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Recursively cautious congestion control," in *Proc. 11th USENIX Symp. NSDI*, Apr. 2014, pp. 373–385. [Online]. Available: https://www.usenix.org/conference/nsdi14/technicalsessions/presentation/mittal

[261] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, *Quick-Start for TCP and IP*, RFC 4782 (Experimental), Internet Engineering Task Force, Jan. 2007.

[262] G. Fairhurst and A. Sathiaseelan, *Quick-Start for the Datagram Congestion Control Protocol (DCCP)*, RFC 5634 (Experimental), Internet Engineering Task Force, Aug. 2009.

[263] P. Sarolahti, M. Allman, and S. Floyd, "Determining an appropriate sending rate over an underutilized network path," *Comput. Netw.*, vol. 51, no. 7, pp. 1815–1832, May 2007.

[264] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth–delay product networks," in *Proc. ACM SIGCOMM*, Pittsburgh, PA, USA, 2002, pp. 89–102.

[265] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the Internet," in *Proc. IWQoS, Lecture Notes in Computer Science*, Passau, Germany, Jun. 2005, pp. 271–285.

[266] L. Eggert and G. Fairhurst, *Unicast UDP Usage Guidelines for Application Designers*, RFC 5405 (Best Current Practice), Internet Engineering Task Force, Nov. 2008.

[267] A. Vainshtein and Y. Stein, *Structure-Agnostic Time Division Multiplexing (TDM) Over Packet (SAToP)*, RFC 4553 (Proposed Standard), Internet Engineering Task Force, Jun. 2006.

[268] I. Järvinen *et al.*, "Effect of competing TCP traffic on interactive real-time communication," in *Proc. Int. Conf. PAM*, 2013, pp. 94–103.

[269] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: A mechanism for background transfers," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 329–343, Dec. 2002.

[270] A. Kuzmanovic and E. Knightly, "TCP-LP: A distributed algorithm for low priority data transfer," in *Proc. IEEE INFOCOM*, 2003, vol. 3, pp. 1691–1701.

[271] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The quest for LEDBAT fairness," in *Proc. IEEE GLOBECOM*, Miami, FL, USA, Dec. 2010, pp. 1–6.

[272] 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 *Proc. ITC*, Sep. 2010, pp. 1–8.

[273] D. Ros and M. Welzl, "Assessing LEDBAT's delay impact," *IEEE Commun. Lett.*, vol. 17, no. 5, pp. 1044–1047, May 2013.

[274] R. Jesup, "Issues with LEDBAT in wide deployment," in *Proc. IETF*, Vancouver, BC, Canada, Jul. 2012, pp. 1–11. [Online]. Available: http://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-2.pdf

[275] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Täht, "Fighting the bufferbloat: On the coexistence of AQM and low priority congestion control," in *Proc. IEEE INFOCOM*, 2013, pp. 3291–3296.

[276] M. Handley, J. Padhye, and S. Floyd, *TCP Congestion Window Validation*, RFC 2861 (Experimental), Internet Engineering Task Force, Jun. 2000.

[277] A. Sathiaseelan, R. Secchi, and G. Fairhurst, "Enhancing TCP to support rate-limited traffic," in *Proc. ACM SIGCOMM Workshop CSWS*, Nice, France, 2012, pp. 39–44.

[278] G. Fairhurst and A. Sathiaseelan, *Updating TCP to Support Rate-Limited Traffic*, Internet Draft draft-ietf-tcpm-newcwv, Mar. 2013, Work in progress.

[279] G. Fairhurst, A. Sathiaseelan, and R. Secchi, *Updating TCP to Support Rate-Limited Traffic*, Internet Draft draft-fairhurst-tcpm-newcwv, Sep. 2012, Work in progress. [Online]. Available: http://tools.ietf.org/html/draft-fairhurst-tcpm-newcwv

[280] S. Floyd, *Limited Slow-Start for TCP With Large Congestion Windows*, RFC 3742 (Experimental), Internet Engineering Task Force, Mar. 2004.

[281] N. Hu and P. Steenkiste, "Improving TCP startup performance using active measurements: Algorithm and evaluation," in *Proc. 11th IEEE ICNP*, Nov. 2003, pp. 107–118.

[282] D. Cavendish, K. Kumazoe, M. Tsuru, Y. Oie, and M. Gerla, "CapStart: An adaptive TCP slow start for high speed networks," in *Proc. Iaria Int. Conf. Evolving Internet*, Los Alamitos, CA, USA, 2009, pp. 15–20.

[283] R. Wang, G. Pau, K. Yamada, M. Y. Sanadidi, and M. Gerla, "TCP start up performance in large bandwidth delay networks," in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004.

[284] S. Giordano, G. Procissi, F. Russo, and R. Secchi, "On the use of pipesize estimators to improve TCP transient behavior," in *Proc. IEEE ICC*, 2005, pp. 16–20.

[285] *The Bufferbloat Projects*, Oct. 2013. [Online]. Available: http://www.bufferbloat.net/

[286] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," *ACM SIGCOMM CCR*, vol. 28, no. 4, pp. 315–323, Oct. 1998.

[287] J. Corbet, *TCP Small Queues*, Linux Weekly News, Jul. 2012. [Online]. Available: https://lwn.net/Articles/507065/

[288] A. Goel, L. Abeni, C. Krasic, J. Snow, and J. Walpole, "Supporting time-sensitive applications on a commodity OS," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 165–180, Dec. 2002.

[289] B. P. Swenson and G. F. Riley, "A new approach to zero-copy message passing with reversible memory allocation in multi-core architectures," in *Proc. IEEE/ACM/SCS 26th Workshop PADS*, 2012, pp. 44–52.

[290] T. Suzumura, M. Tatsubori, S. Trent, A. Tozawa, and T. Onodera, "Highly scalable web applications with zerocopy data transfer," in *Proc. ACM Int. Conf. World Wide Web*, Madrid, Spain, 2009, pp. 921–930.

[291] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion control without reliability," *ACM SIGCOMM CCR*, vol. 36, no. 4, pp. 27–38, Aug. 2006.

[292] M. Welzl, S. Jorer, and S. Gjessing, "Towards a protocol-independent Internet transport API," in *Proc. IEEE ICC*, 2011, pp. 1–6.

[293] L. Eggert and W. M. Eddy, "Towards more expressive transport-layer interfaces," in *Proc. ACM Workshop MobiArch*, San Francisco, CA, USA, 2006, pp. 71–74.

[294] A. Petlund, "Transport Services and Low Latency," Internet Draft draft-petlund-latency-transport-services, Feb. 2014, Work in progress. [Online]. Available: http://tools.ietf.org/html/draft-petlund-latency-transport-services

[295] G. De Micheli, R. Ernst, and W. Wolf, Eds., *Readings in Hardware/Software Co-Design*. Norwell, MA, USA: Kluwer, 2002.

[296] D. E. Culler, A. Gupta, and J. P. Singh, *Parallel Computer Architecture: A Hardware/Software Approach*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann, 1997.

[297] D. A. Patterson and J. L. Hennessy, "Computer Organization and Design, The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)," San Francisco, CA, USA: Morgan Kaufmann, 2008.

[298] R. H. B. Netzer and B. P. Miller, "What are race conditions?: Some issues and formalizations," *ACM Lett. Program. Languages Syst.*, vol. 1, no. 1, pp. 74–88, Mar. 1992.

[299] S. Carr, J. Mayo, and C.-K. Shene, "Race conditions: A case study," *J. Comput. Sci. Colleges*, vol. 17, no. 1, pp. 90–105, Oct. 2001.

[300] U. Cummings and M. Zeile, "FocalPoint II, a low-latency, high bandwidth switch/router chip," in *Proc. Symp. Hot Chips*, Stanford, CA, USA, Aug. 2007, pp. 1–21. [Online]. Available: http://www.hotchips.org/wpcontent/uploads/hc_archives/hc19/3_Tues/HC19.07/HC19.07.03.pdf

[301] A. I. C. Grecu *et al.*, "Towards open network-on-chip benchmarks," in *Proc. IEEE/ACM Int. Symp. NOCS*, Princeton, NJ, USA, May 2007, p. 205. [Online]. Available: http://web.it.kth.se/~axel/papers/2007/NOCS-Benchmarks.pdf

[302] G. P. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan, "On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects," *ACM SIGCOMM CCR*, vol. 42, no. 4, pp. 407–418, Aug. 2012.

[303] J. Owens *et al.*, "Research challenges for onchip interconnection networks," *IEEE Micro*, vol. 27, no. 5, pp. 96–108, Sep./Oct. 2007.

[304] M. Ali, M. Welzl, and M. Zwicknagl, "Networks on chips: Scalable interconnects for future systems on chips," in *Proc. 4th Eur. Conf. Circuits Syst. Commun.*, Jul. 2008, pp. 240–245.

[305] S. Rumble, D. Ongaro, S. Stutsman, M. Rosenblum, and J. Ousterhout, "It's time for low latency," in *Proc. 13th USENIX Workshop HotOS*, Napa Valley, CA, USA, May 2011, p. 11. [Online]. Available: https://www.usenix.org/legacy/events/hotos11/tech/final_files/Rumble.pdf

[306] "SPDY: An Experimental Protocol for a Faster Web," White paper. [Online]. Available: http://dev.chromium.org/spdy/spdy-whitepaper
[307] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY really make the web faster?" in *Proc. IFIP Netw.*, Jun. 2014, pp. 1–9.
[308] J. Roskind, *QUIC: Quick UDP Internet Connections, (Living Document)*, Dec. 2013. [Online]. Available: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFVev2jRFUoVD34/edit
[309] A. Langley and W. T. Chang, *QUIC Crypto*, Mar. 2014. [Online]. Available: https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit
[310] Google, *The Chromium Project*, Mountain View, CA, USA. [Online]. Available: http://www.chromium.org/Home
[311] Connectify, *Taking Google's QUIC for a Test Drive*, Philadelphia, PA, USA. Nov. 2013. [Online]. Available: http://www.connectify.me/taking-google-quic-for-a-test-drive/
[312] Riverbed, *Riverbed Optimization System (RiOS) 6.1; A Technical Overview*, San Francisco, CA, USA. 2010. [Online]. Available: http://www.riverbed.com/about/document-repository/riverbed-optimization-system-rios-version-61.html
[313] T. Grenot, F. Bonnet, B. Imbert, and J. Provost, "Method and device for remotely controlling the congestion of meshed flow in a packet mode telecommunication network," U.S. Patent 20 080304 414 A1, Dec. 11, 2008.
[314] T. Grenot, "Method for optimising the sharing of a plurality of network resources between a plurality of application flows," U.S. Patent 2 673 325 A1, Jun. 26, 2010.
[315] M. Delattre and B. Imbert, "Method for management of data stream exchanges in an autonomic telecommunications network," U.S. Patent 2 012 0023 213 A1, Jan. 26, 2012.
[316] G. Henke, *Acceleration; Bottlenecks, Pitfalls and Tips*, White Paper. [Online]. Available: http://cdn.crn.de/fileadmin/whitepapers/files/ipanema_acceleration_bottleneck_pitfalls_tips.pdf
[317] Ipanema, *Guarantee Applications Performance With Dynamic Hybrid Networking*, White Paper, 2013. [Online]. Available: http://response.ipanematech.com/WP_Dynamic_Hybrid_Networking_EN?i=165
[318] C. D. Sutton, "Using QoS tunnels for TCP latency optimization," U.S. Patent 20 080 069 111 A1, Mar. 28, 2008.
[319] Infinita, *Overcoming the Limits of TCP on High-Speed WANs*, White Paper, 2011. [Online]. Available: http://www.dabcc.com/downloadfile.aspx?id=1162
[320] J. Pinkerton, *SMB2—Big Improvements in the Remote Filesystem Protocol*, SNIA, Online Educational Resource, San Jose, CA, USA, 2008. [Online]. Available: http://www.snia.org/sites/default/education/tutorials/2008/fall/networking/JimPinkerton-SMB2_Big_Improvements_Remote_FS_Protocolv3.pdf
[321] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations*, RFC 3135 (Informational), Internet Engineering Task Force, Jun. 2001.
[322] J. Touch, "Two Ways to Trade Bandwidth for Latency," in *Proc. ISOC Workshop Reducing Internet Latency*, Sep. 2013, pp. 1–2. [Online]. Available: http://www.internetsociety.org/sites/default/files/pdf/accepted/11_touch-isoc-latency-2013.pdf
[323] M. Chetty, S. Sundaresan, S. Muckaden, N. Feamster, and E. Calandro, "Measuring broadband performance in South Africa," in *Proc. 4th ACM Annu. Symp. DEV*, 2013, pp. 1–10. [Online]. Available: http://sites.noise.gatech.edu/~srikanth/docs/broadband-sa-dev4.pdf
[324] *DNS Prefetching (or Pre-Resolving)*, Jan. 2014. [Online]. Available: http://blog.chromium.org/2008/09/dns-prefetching-or-pre-resolving.html
[325] T. Everts, *11 Questions (and answers) about content delivery networks and web performance*, Jan. 2014. [Online]. Available: http://www.webperformancetoday.com/2013/06/12/11-faqs-content-delivery-networks-cdn-web-performance/
[326] I. Rhee, L. Xu, and S. Ha, "CUBIC for Fast Long-Distance Networks," Internet Draft draft-rhee-tcpm-cubic, Aug. 2008. [Online]. Available: http://tools.ietf.org/html/draft-rhee-tcpm-cubic
[327] J. Vardakas, I. Papapanagiotou, M. Logothetis, and S. Kotsopoulos, "On the end-to-end delay analysis of the IEEE 802.11 distributed coordination function," in *Proc. 2nd ICIMP*, Jul. 2007, p. 16.
[328] T. Selvam and S. Srikanth, "A frame aggregation scheduler for IEEE 802.11n," in *Proc. NCC*, Jan. 2010, pp. 1–5.
[329] B. Ginzburg and A. Kesselman, "Performance analysis of A-MPDU and A-MSDU aggregation in IEEE 802.11n," in *Proc. of the IEEE Sarnoff Symposium*, Apr. 2007, pp. 1–5.
[330] G. Bhanage, R. Mahindra, I. Seskar, and D. Raychaudhuri, "Implication of MAC frame aggregation on empirical wireless experimentation," in *Proc. IEEE GLOBECOM*, Nov. 2009, pp. 1–7.
[331] S. Gubner and C. Lindemann, "Evaluating the impact of frame aggregation on video-streaming over IEEE 802.11n multihop networks," in *Proc. of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Jun. 2012, pp. 1–6.
[332] S. Sundaresan *et al.*, "Broadband Internet performance: A view from the gateway," in *Proc. ACM SIGCOMM*, Toronto, ON, Canada, 2011, pp. 134–145. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018452
[333] D. Newman, "Latency and jitter: Cut-through design pays off for arista, blade," IDG Network World, Framingham, MA, USA, Jan. 2010. [Online]. Available: http://www.networkworld.com/reviews/2010/011810-ethernet-switch-test-latency.html
[334] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of red's active queue management," ICIR, Technical report, Frankfurt, Germany, Aug. 2001. [Online]. Available: http://www.icir.org/floyd/red.html
[335] B. Briscoe, "Review: Quick-Start for TCP and IP," BT, Ipswich, U.K., Tech. Rep. TR-CXR9-2005-007, Nov. 2005. [Online]. Available: http://www.bobbriscoe.net/projects/2020comms/refb/jain05_rvw_rjb.pdf
[336] I. Biswas, A. Sathiaseelan, R. Secchi, and G. Fairhurst, "Analysing TCP for bursty traffic," *Indersci. IJAMC*, vol. 7, no. 3, pp. 585–592, Jun. 2010.
[337] A. Angelogiannopoulos, "Implementation and performance evaluation of TCP extensions in FreeBSD," M.S thesis, Technische Universität München Lehrstuhl für Kommunikationsnetze, Munchen, Germany, Oct., 2013. [Online]. Available: https://eggert.org/students/angelogiannopoulos-thesis.pdf
[338] E. Jeong *et al.*, "mTCP: A highly scalable user-level TCP stack for multicore systems," in *Proc. 11th USENIX Symp. NSDI*, Apr. 2014, pp. 489–502. [Online]. Available: https://www.usenix.org/conference/nsdi14/technicalsessions/presentation/jeong
[339] B. Briscoe *et al.*, "A survey of latency reducing techniques and their merits," in *Proc. ISOC Workshop Reducing Internet Latency*, Sep. 2013, pp. 1–2. [Online]. Available: http://www.Internetsociety.org/sites/default/files/pdf/accepted/16_rite-latency_survey_pos.pdf
[340] J. P. G. Sterbenz and J. D. Touch, *High-Speed Networking—A Systematic Approach to High-Bandwidth Low-Latency Communication*. Hoboken, NJ, USA: Wiley, 2001.

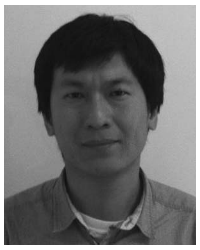
Bob Briscoe attained a PhD in 2009 from University College London, U.K. He is the Chief Researcher in Network Infrastructure at BT, Ipswich, U.K., and a member of BT's Network Strategy team. His expertise is in engineering and economic and social control of computer networks. In the late 1980s, he managed the transition to IP of many of BT's R&D Labs. In 2000, he set up and led the Market Managed Multi-service Internet (M3I) consortium and incubated a start-up that BT absorbed into its Internet QoS products. In 2007 he helped initiate the Trilogy project, which successfully delivered re-definition of the Internet architecture through IETF standardization. His recent work is on reducing latency and on virtualizing network functions (NFV), particularly security aspects.

Anna Brunstrom received the B.Sc. degree in Computer Science and Mathematics from Pepperdine University, Malibu, CA, USA, in 1991 and the M.Sc. and Ph.D. degrees in Computer Science from the College of William and Mary, Williamsburg, VA, USA, in 1993 and 1996, respectively. She joined the Department of Computer Science, Karlstad University, Sweden, in 1996, where she is currently a Full Professor and the Research Manager for the Distributed Systems and Communications Research Group. She has a background in distributed systems, but her main area of work over the last years has been in computer networking with a focus on transport protocol design, QoS/QoE issues, cross-layer interactions, and wireless communication. She has authored/coauthored ten book chapters and over 100 international journal and conference papers.

Andreas Petlund received the Ph.D. degree from the University of Oslo, Norway, in 2009. His Ph.D. thesis focused on low-latency transport for interactive and time-dependant applications. He is currently a Research Scientist at Simula Research Laboratory, Fornebu, Norway. He is currently coordinating the "Reducing Internet Transport Latency" (RITE) FP7 EU project and leading a national project on low latency for thin-stream applications. His work on retransmission mechanisms to reduce latency has resulted in a suite of mechanisms, several of which are available in the Linux kernel. The main topic of his current work is systems and network optimizations for time-dependant applications, but he also has experience in kernel-level optimizations, embedded systems, and heterogeneous processor systems.

David Hayes received the BE(Elect) degree from Queensland University of Technology, Australia, in 1987 and his PhD from the University of Melbourne, Australia, in 2002. Initially, he worked with Telstra (Australia) and then with Ngee Ann Polytechnic (Singapore). Since then, he has had a number of positions, including those with the University of Melbourne, Queensland University of Technology, Swinburne University of Technology (Australia), and his current position at the University of Oslo, Norway, working on the "Reducing Internet Transport Latency" (RITE) FP7 EU project. He has authored delay-based TCP congestion controls and SCTP NAT in FreeBSD. He has interests in various aspects of network performance research, analysis, and protocol implementation.

David Ros received his B.Sc. (with honors) and M.Sc. degrees in Electronics Engineering from Simón Bolívar University, Caracas, Venezuela, and his Ph.D. in Computer Science from the Institut National de Sciences Appliquées, Rennes, France. After a long tenure as an Associate Professor in Télécom Bretagne's Networks, Security and Multimedia Department, he moved to Simula Research Laboratory, Fornebu, Norway, where he is currently working as Coordinator of EU Research for the Section in Communication Systems. He is currently co-chairing the Internet Congestion Control Research Group at the IRTF. His active research interests include transport-layer issues, congestion control, as well as quality of service and architectural issues in IP networks.



Ing-Jyh Tsang received the B.Sc. degree in Electronic Engineering from the Federal University of Pernambuco, Recife, Brazil, and the Ph.D. degree in Physics from the University of Antwerp, Antwerp, Belgium. He joined Alcatel-Lucent, Antwerpen, in 2000, starting at the former Research and Innovation Department working on BPON/GPON and IPTV services. He worked in several departments, as System Engineer at Wireline Division, Solution Architect within the Service Routing Department, and Consultant Network Architect within a major operator. He is a Senior Research Engineer at Bell Labs—Network Algorithms, Protocols and Security (NAPS) group, and having participated in several EU funded projects such as GIANT and ECODE, at present, he is working on the "Reducing Internet Transport Latency" (RITE) FP7 EU project.

Stein Gjessing received the Cand. Real. and Dr. Philos. degrees from the University of Oslo, Oslo, Norway, in 1975 and 1985, respectively. He is a Professor of computer science with the Department of Informatics, University of Oslo. He acted as Head of the Department of Informatics for four years from 1987. From February 1996 to October 2001, he was the Chairman of the national research program "Distributed IT-System," founded by the Research Council of Norway. His original work was in the field of programming languages and programming language semantics, in particular related to object oriented concurrent programming. He has worked with computer interconnects and computer architecture for cache coherent shared memory, with DRAM organization, with ring based LANs (IEEE Standard 802.17) and with IP fast reroute. His current research interests are transport, routing, and network resilience both in Internet-like networks and in sensor networks.

Gorry Fairhurst received the B.Sc. degree in Applied Physics and Electronics from Durham University, Durham, U.K., and the Ph.D. degree in Communications Engineering from the University of Aberdeen, Aberdeen, U.K. He is currently a Professor with the School of Engineering, University of Aberdeen. His research interests include link communications protocols, TCP transport, development of multicast transport protocols, networking techniques for low latency Internet communication, and performance evaluation of broadband satellite systems. He has worked on a range of IP-based satellite projects funded by national, European, and ESA funding, and contributed to DVB on networking standards for IP transmission over DVB and the HLS for DVB-RCS2. He actively participates in developing networking standards with the Internet Engineering Task Force, where he chairs the Transport and Services Working Group (TSVWG) and is a member of the IETF Transport Directorate.

Carsten Griwodz received the Diploma in Computer Science from the University of Paderborn, Paderborn, Germany, in 1993 and the Ph.D. degree from Darmstadt University of Technology, Germany, in 2000. From 1993 to 1997, he was with the IBM European Networking Center, Heidelberg, Germany. In 1997, he joined the Multimedia Communications Laboratory, Darmstadt University of Technology. He joined the University of Oslo, Norway, in 2000 and the research company Simula Research Laboratory, Fornebu, Norway, in 2005. He has been a Full Professor at the University of Oslo since 2006 and has led the Media Department, Simula Research Laboratory, since 2009. His research interest is the performance of multimedia systems. He is concerned with streaming media, which includes all kinds of media that are transported over the Internet with temporal demands, including stored and live video as well as games and immersive systems.

Michael Welzl received Ph.D. (with distinction) and Habilitation degrees from Darmstadt University of Technology, Germany, in 2002 and 2007 respectively. He spent two years as a Research Assistant at the Telecooperation Department, University of Linz/Austria, before joining the faculty of the newly founded Institute of Computer Science, University of Innsbruck/Austria, in November 2001, where he led a research team on Network Support for Grid Computing. In May 2009, he joined the Department of Informatics, University of Oslo, Oslo, Norway, as an Associate Professor. He was appointed to a Full Professorship in September 2009. Since October 2011, he has also been an Adjunct Professor at Swinburne University of Technology, Melbourne, Australia.