TCP Hollywood: An Unordered, Time-Lined, TCP for Networked Multimedia Applications

Stephen McQuistin
University of Glasgow, UK
sm@smcquistin.uk

Colin Perkins
University of Glasgow, UK
csp@csperkins.org

Marwan Fayed
University of Stirling, UK
mmf@cs.stir.ac.uk

***Abstract*—Ossification of the transport-layer limits networked multimedia applications to use TCP or UDP, despite standardisation of new transport protocols that better support their requirements. To improve transport for these applications, we present TCP Hollywood, an unordered, time-lined, TCP variant designed to support real-time multimedia traffic while being widely deployable. Analysis of the protocol indicates that it increases the utility of the network in lossy conditions where total one-way delay is constrained, such as with telephony applications and low-latency video streaming. This allows retransmissions to be useful in cases where they are not with standard TCP, improving the timely good-put of the protocol and reducing overheads. Initial experiments show that TCP Hollywood is deployable on the Internet, successfully operating on all major fixed and mobile networks in the UK, with safe failure modes.**

I. Introduction

Real-time networked multimedia applications have long contributed to Internet traffic. This can take the form of telephony [1], video conferencing [2], live or on-demand TV and movies [3], [4], or user-generated video. These applications, and the traffic they generate, are rapidly increasing in popularity, and now comprise the majority of Internet traffic [5].

The nature of such real-time traffic is that it prefers predictable and bounded latency to strict reliability, since data that arrives too late is as bad as data that does not arrive at all. This suggests that data should be sent in packets that can be independently decoded [6], to allow them to be processed irrespective of the loss or delay of other packets. However, the requirement for efficient media compression leads to interdependence between packet contents and codecs that operate across multiple frames. When coupled with challenging network environments, such as mobile wireless, that have unreliable delivery and unpredictable latency, the requirements for effective media transport become difficult to satisfy.

Applications access the network via the transport layer. The transport protocol should provide services to meet the application demands, abstracting away details of the transport process, and delivering data with an appropriate degree of reliability and timeliness. For real-time networked multimedia, the transport should be trusted to minimize transport-induced delay, and should respect (partial) reliability semantics pertaining to media importance, deadlines, and dependencies.

Message-oriented transports, such as SCTP [7] and DCCP [8], ought to be suitable building blocks, but their deployment is restricted by NATs, firewalls, and other middleboxes [9]. This leaves real-time applications to use UDP or TCP, neither of which is well-suited to their needs. UDP contributes minimal latency, making it the recommended transport to meet the strict latency bounds of real-time applications [10], but provides limited support to applications, and is commonly blocked by enterprise firewalls. TCP prefers reliability to timeliness, and its congestion control tends to drive up queueing delay, but is often the only transport that can pass through middleboxes on the path. Accordingly, and despite its many problems, TCP is rapidly becoming the de facto transport for multimedia traffic.

ISBN 978-3-901882-83-8 © 2016 IFIP

In this paper, we engineer TCP Hollywood in response to these trends. TCP Hollywood is an unordered and time-lined transport protocol, that is wire compatible with standard TCP, but eliminates two sources of transport-induced latency, and provides reliability semantics that better suit real-time multimedia applications. Specifically, TCP Hollywood: 1) removes head-of-line blocking at the receiver and delivers received data to the application immediately, irrespective of ordering; and 2) relaxes reliability to respect time lines provided by the application, so only data that will arrive in time is retransmitted, otherwise retransmissions carry new data. The combination of both design elements reduces latency and introduces message-oriented semantics, allowing TCP Hollywood to express inter-dependencies between messages. Crucially, TCP Hollywood is wire-compatible with TCP, and incrementally deployable on the public Internet.

Our implementation consists of an intermediate logic layer that sits between the application and the kernel. Extensions in the TCP stack facilitate out-of-order delivery, and can be enabled or disabled via socket options. Messages are delineated in the logic layer using timing and dependency information from the application, and COBS-encoded [11] to survive re-segmentation that may occur in the network. We introduce the concept of inconsistent retransmissions: if the round-trip time (RTT) estimator indicates that a message will arrive too late to be useful, or if a message depends on a previous unsuccessfully transmitted message, then TCP Hollywood can exploit re-transmission slots to send new data and avoid retransmitting useless data. The semantics of TCP are maintained by preserving the sequence numbers in retransmitted segments, whether inconsistent or not. We develop an analytical framework to model the value of a retransmission against the buffering and processing time of data at the receiver-side. Our analysis reveals a wide range of RTT values where standard TCP retransmissions will arrive too late to be useful. We use this model to validate TCP Hollywood, and show that it handles retransmissions correctly.

Our contributions are as follows. After reviewing the rationale and requirements in Section II, we design a TCP-compatible architecture and application programming interface (API) that eliminates transport related, but not congestion control related, delay from TCP in Section III (this can be used with any of the existing proposals to reduce congestion control related delay, such as active queue management [12], [13] or delay-based congestion control [14], [15]). We develop an analytic framework in Section IV to determine the value and content of retransmitted data. We outline our implementation in Section V, alongside experiments to demonstrate ease of deployment. Related work and concluding remarks are provided in Sections VI and VII, respectively.

II. Rationale and Requirements

We begin by considering in more detail the requirements and rationale for an unordered and time-lined transport protocol. It is instructive to establish TCP as the foundation from which to build, and understand its negative impact in the context of live and interactive media.

Our primary design goal is to improve performance over TCP *for real-time traffic*, while maintaining deployability on the scale of TCP and UDP. Ossification of the transport layer means this goal can only be achieved by using TCP or UDP as a substrate. This is a limitation that exists in the Internet because of middleboxes that process packets based on static views of what is a valid transport. The operation of these middleboxes places two constraints on transport protocols [16]. First, only payloads marked as TCP or UDP are marked as valid; payloads carried by other transport protocols are often rejected. Second, middleboxes may reject valid TCP packets that don't conform to some limited subset of the TCP protocol that is understood by the middlebox [17]. For example, packets with the SACK (selective acknowledgement) extension might be rejected by a middlebox that doesn't understand that extension, and expects only regular ACK packets. In this restricted domain, reliability and congestion control are desirable features, that are difficult to implement at the application layer, and have forced TCP to emerge as the protocol of choice for real-time multimedia, despite struggling to meet latency bounds.

Our secondary goal is to minimize the transport-induced latency on applications. With TCP selected as the substrate, it remains to determine the appropriate modifications to meet our latency goals. TCP introduces latency in part because of the nature of its congestion control dynamics, and in part by providing an ordered, reliable, delivery model using head-of-line blocking and retransmissions. The former can be addressed using active queue management and/or delay-based congestion control algorithms, and has been widely studied. The latter issue is more applicable for real-time traffic, and is the subject of our work. Figure 1 shows the impact of head-of-line blocking: The loss of the third segment causes subsequent segments to be buffered at the receiver while waiting for the retransmission. Only when the delayed segment arrives can TCP deliver the in-order sequence to the application. The impact of retransmissions are exacerbated when they push segments outside of the window in which they are useful to the application. Effectively such segments are lost to the application, despite having been delivered to the host on time. It is these *late losses* that TCP Hollywood seeks to minimize.

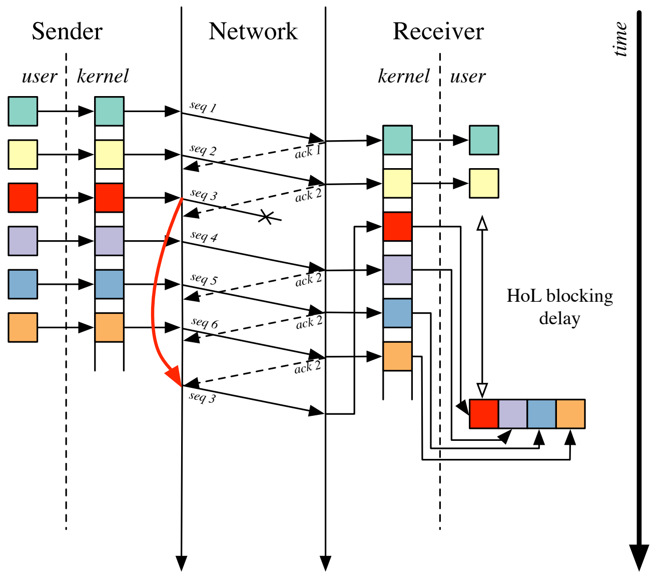


Figure 1. The interaction between head-of-line blocking and loss in TCP: multiple segments are delayed by a loss, and potentially delivered too late to be useful to the receiver

Two requirements follow. First, segments must be delivered as they arrive to eliminate head-of-line blocking. Second, retransmissions should be evaluated against timing information to ensure the delivery of useful data, by allowing *inconsistent retransmissions* to send new data in a segment that is retransmitted. So that applications can benefit from out-of-order delivery, a message-oriented abstraction is needed. Specifically, messages should be independently useful to the receiver [6]. With both a message-oriented abstraction and timing information, the collection of message dependency information follows. This increases the application-awareness of the transport layer.

III. Architecture and Design

TCP Hollywood has been designed to be deployable on the ossified Internet as it exists today, and to support partial deployments where only the sender or the receiver has been upgraded to support the TCP Hollywood extensions. The nature of the extensions we propose supports the former, while the latter is achieved by splitting the functionality between a user-space intermediary 'shim' layer and a set of extensions to the kernel TCP stack. The intermediary layer operates over either unmodified TCP, or with the TCP Hollywood kernel extensions enabled. The user and kernel components are represented in the overall architecture represented in Figure 2. In discussing the architecture it is useful to consider the sender separately from the receiver, and for each to consider the user-space intermediary layer separately from the kernel TCP extensions.

A. *TCP Hollywood Sender architecture*

The architecture of a TCP Hollywood sender is shown in the left hand side of Figure 2. The sender inherits the requirements identified in Section II to support a timed, message-oriented, transport abstraction, with inconsistent retransmissions.

The intermediary layer provides the message-oriented abstraction. It accepts a sequence of messages (i.e., datagrams,

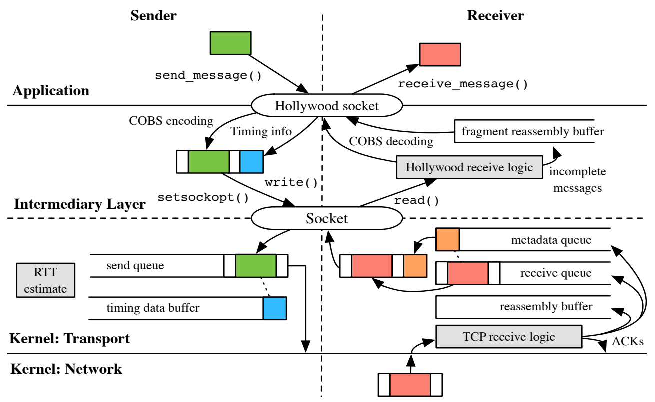


Figure 2. TCP Hollywood sender and receiver architecture

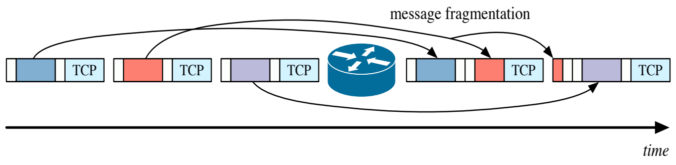


Figure 3. Encoding and framing with leading and trailing markers protects against middlebox re-segmentation; received segments can be properly decoded

rather than a byte stream) from the application, with optional timeliness and dependency information, to be delivered to the destination. The intermediary layer supports a sub-stream abstraction, allowing messages from multiple flows to be multiplexed on a single transport-level connection (similar to how multiple streams can be sent within a single SCTP association [7]). This can be used to cleanly multiplex audio and video flows onto a single connection, or to distinguish multiple layers of a stream encoded using scalable video coding [18], for example using H.264/SVC. The intermediary layer appends a sub-stream identifier to messages before they are encoded, framed, and passed to the kernel TCP sender, with a default sub-stream being reserved for flows where no sub-stream is specified. The application can provide timing or dependency data via the intermediary layer API. This is passed to the kernel alongside the encoded message, and used to determine whether inconsistent retransmissions are appropriate.

To support a message-oriented abstraction over a TCP byte stream, the TCP Hollywood flows must be resilient to re-segmentation or segment coalescing by middleboxes. Message integrity must be protected: messages received must have been sent, and only complete messages must be delivered. This is ensured by the intermediary layer, which frames messages with a leading and trailing marker. The effect is shown in Figure 3, where markers can be used to delineate messages irrespective of the segmentation. The intermediary layer encodes messages with consistent overhead byte stuffing (COBS) [19]; this efficiently encodes the stream to escape all zero bytes, allowing their use as framing markers, while still providing a transparent channel that can carry any message.

The TCP sender implementation in the kernel is modified to perform consistent segmentation, and to manage inconsistent retransmission by tracking message timing, deadline expiration, and dependencies. Consistent segmentation ensures that a single `write()` call made by the intermediary layer will generate a single TCP segment, provided the size of the segment does not exceed the MTU. This ensures each message is sent in a separate TCP segment, allowing the receiver to process it independently of other messages, reducing latency. This implies disabling Nagle's algorithm (i.e., setting the `TCP_NODELAY` socket option) to avoid unnecessary buffering – Nagle's algorithm would not provide a significant benefit to our target applications, where messages are large compared to their headers.

TCP retransmissions ensure reliability, but also inject latency that may cause late losses. A TCP Hollywood sender has the notion of *message expiry*: a message expires when (i) RTT estimates indicate the retransmitted message will arrive too late, or (ii) if the message depends on a previous message that was unsuccessfully delivered. Under these circumstances TCP Hollywood can send a new message using the same TCP sequence number space as a previously sent message, re-writing the remaining bytes in the TCP send buffer with new content. To support such *inconsistent retransmissions*, the intermediary layer passes messages down to the modified kernel TCP stack along with metadata to describe their deadline, dependency, and sub-stream. This is enabled by calls to the Berkeley Sockets API `setsockopt()` function. The metadata, with the exception of the sub-stream identifier, is never transmitted on the wire, but is held locally for each message for as long as the message is buffered (i.e., until all `ACK`s associated with a message are received). Our kernel extensions implement a separate buffer to hold per-message metadata.

The inconsistent retransmission logic is triggered when the standard TCP retransmission logic would be triggered by a triple duplicate ACK or timeout. Metadata for unacknowledged messages is then evaluated against the current RTT estimate, to determine whether the original message is to be retransmitted, or if an inconsistent retransmission is to be sent, replacing the original data with new content while keeping the same TCP sequence number. Since messages are framed and self-describing, a receiver can decode the inconsistent retransmission.

The latency benefits of inconsistent retransmissions will be quantified in Section IV. In the interim, we emphasize the message abstraction in this context: TCP Hollywood sends messages rather than bytes in a data stream. Consequently, a message may be composed of multiple fragments, split across TCP segments. To preserve the semantics at the receiver, fragments necessary to finish a partially received message are always retransmitted, but if no part of a message was received, it may be replaced with a new message when its containing TCP segment is retransmitted.

The processing overhead of TCP Hollywood at the sender is comprised of COBS encoding at the intermediary layer, and the maintenance of metadata in the kernel. COBS encoding requires a copy of the message to be made, but this could be eliminated by performing the byte stuffing as the message is being generated, as part of the multimedia encoding. Beyond this copy, COBS is "computationally cheap" [11]. In the kernel, the sender maintains metadata for each message, while the message could still be sent. Further processing, such as estimating whether a message will arrive on time, uses data already maintained by the kernel.

B. *TCP Hollywood Receiver Architecture*

The receiver-side architecture of TCP Hollywood is shown on the right hand side of Figure 2. Like the sender, it is composed of a user-space intermediary layer, and TCP extensions in the kernel receive path. The receiver supports message-oriented delivery, and additionally eliminates head-of-line blocking. The use of inconsistent retransmissions is invisible to the receiver.

The kernel initially processes incoming segments as would TCP. It generates the appropriate ACKs (e.g., duplicate ACKs for out-of-order or lost segments), and places segments into the reassembly buffer as usual. The on-the-wire response to each received segment is *identical* to that of TCP: ACKs (and SACK blocks, or other extensions, if negotiated) are generated in exactly the same way as standard TCP, and the congestion response is unchanged.

Where a TCP Hollywood receiver differs from standard TCP is that all segments, including those received out-of-order, are delivered to the intermediary layer in the order they are received, with no head-of-line blocking or reordering. As each segment arrives, a metadata structure is created to store its TCP sequence number. This sequence number is then appended to the segment as it is read by the intermediary layer. Sequence numbers are used by the intermediary layer to delineate messages that are encoded across multiple segments. Making segments available to the intermediary layer as they arrive is the only change needed to the kernel TCP code at the receiver.

The intermediary layer scans incoming segments for complete messages, delineated by the COBS framing. If consistent segmentation was used, and segments were not fragmented or coalesced in the network, then messages will correspond to TCP segments. Otherwise, incomplete message fragments are buffered in the fragment reassembly buffer awaiting missing fragments. The relative ordering of the bytes in message fragments is established using the TCP sequence number tag associated with received segments. As shown in Figure 2, complete messages are decoded and queued for delivery to the application. The API between intermediary layer and application is message oriented, and includes a message sequence number. This simplifies receiver processing compared to the TCP stream API.

The COBS decoding process is similar to that of the receiver, incurring an additional copy at the intermediary layer. In the kernel, our proof-of-concept implementation maintains a metadata structure to store the TCP sequence number and length of each incoming segment – data that would be otherwise lost. For incoming segments that are out-of-order, or arrive while there are segments in the reassembly queue, we make an additional copy (versus the TCP implementation) of the segment's payload, storing this with the segment's metadata. While this simplifies the implementation, it is not a requirement of the design: optimisation of our implementation could eliminate this.

C. *Partial Deployments and Legacy TCP Compatibility*

The TCP Hollywood intermediary layer is a user-space library that can run over a standard TCP implementation, using the Berkeley Sockets API, or on a modified TCP stack using the extensions we have described. If both sender and receiver support the kernel TCP extensions, the full benefit described above is achieved. However, the TCP Hollywood intermediary layer can also be deployed as part of an application, irrespective of the state of deployment of the kernel TCP extensions.

If only the receiver supports the TCP Hollywood kernel extensions, with a standard TCP sender, then the intermediary layer and application will benefit from avoidance of head-of-line blocking, but not from the latency reduction of inconsistent retransmission. Message oriented delivery will be supported, since COBS framing is generated by the intermediary layer at the sender, but COBS decoding may be less efficient since messages boundaries will be less likely to be aligned with segment boundaries.

If only the sender supports the TCP Hollywood kernel extensions, it will generate inconsistent retransmissions, and perform consistent segmentation as described, since both are invisible to the TCP layer of the receiver (compatibility with middleboxes is discussed in Section V-B). This will improve latency, and increase efficiency of COBS decoding, at the receiver, irrespective of whether the receiver has the TCP Hollywood kernel extensions.

If neither sender or receiver support the TCP Hollywood kernel extensions, the intermediary layers can communicate over a standard TCP connection. In this case, the message oriented abstraction persists, and applications can communicate using a TCP Hollywood socket to exchange messages, rather than byte streams, in a congestion controlled and reliable manner, although with no latency benefit over standard TCP.

IV. Latency Reductions and Analysis

TCP Hollywood reduces transport latency through support of inconsistent retransmissions, and by eliminating receiver-side head-of-line blocking. To quantify the benefits of these two techniques to the application, we begin by modelling the one-way transport delay, T_{owd}, as:

$$T_{\text{owd}} = T_{\text{sender}} + T_{\text{playout}} + T_{\text{rtt}}/2 \tag{1}$$

where T_{sender} is the time taken for the sender to capture, encode, and transmit a frame of media data. T_{playout} is the sum of the de-jitter buffering delay, and the time taken to decode and render a frame to the application at the receiver. Finally, T_{rtt} is the network round-trip time. With no loss of generality we assume broadly symmetric network paths in this analysis.[1]

The inter-frame interval of the media, i.e., the duration of media in each frame, is denoted by T_{framing}. We know that $T_{\text{sender}} \geq T_{\text{framing}}$, since a frame cannot be sent before it has been captured. Similarly at the receiver, if the media is to be decoded and rendered without gaps, then $T_{\text{playout}} \geq T_{\text{framing}}$. The time needed to encode and decode media is generally negligible in comparison to the framing interval, making $T_{\text{sender}} \approx T_{\text{playout}}$ a reasonable approximation in the absence of jitter. At the receiver, however, while the media decoding and rendering

[1]This assumption does not hold in ADSL and cellular networks with asymmetric downstream and upstream links. In these cases, our model mis-approximates the application's upper bound on delay, shifting the line marked "Application Deadline" in Figure 4. While further analysis is needed to quantify the impact of this, it is clear that it does not change the broad conclusion of our analysis: that TCP Hollywood increases the usable region of retransmissions.

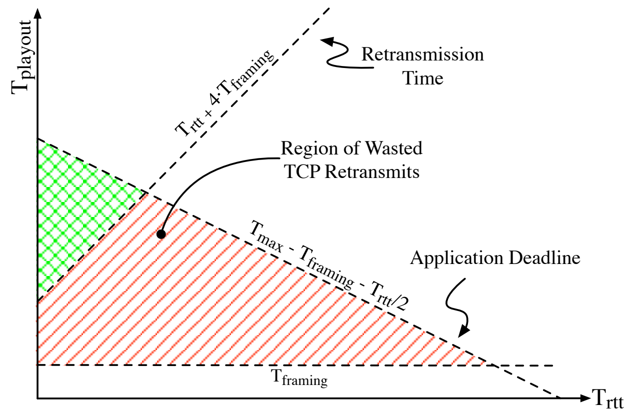

$T_{playout}$
Retransmission Time
$T_{rtt} + 4 \cdot T_{framing}$
Region of Wasted TCP Retransmits
$T_{max} - T_{framing} - T_{rtt}/2$
Application Deadline
$T_{framing}$
T_{rtt}

Figure 4. *Inconsistent Retransmissions* for real-time applications: TCP retransmissions may arrive too late to be used, if the play-out delay is set to meet the application deadline

time is generally small, the de-jitter buffer duration can be significant, and a similar approximation cannot be made.

The one-way transport delay contributes to an application's acceptable delay bound T_{max}, such that $T_{\text{owd}} \leq T_{\text{max}}$. For interactive applications, the delay bound is generally around 150ms [20], whereas streaming applications can accept longer delay bounds (around 0.5 seconds if channel surfing is to be supported; up to tens of seconds for on-demand streaming).

A. Utility of Inconsistent Retransmissions

TCP senders interpret a triple duplicate acknowledgement as an indication of packet loss, and retransmit the missing packet. It follows that the time needed by a sender to identify packet loss following a transmission has a lower bound of:

$$T_{\text{rexmit}} = T_{\text{rtt}} + 3 \times T_{\text{framing}} \tag{2}$$

At the receiver there is one additional framing interval to compensate for the interval that was lost with the original transmission. Assume media decoding and rendering take a negligible time. A retransmitted packet will arrive in time to be received and rendered to the application, provided:

$$T_{\text{playout}} \geq T_{\text{rexmit}} + T_{\text{framing}} \tag{3}$$

When $T_{\text{playout}} < T_{\text{rexmit}}$, retransmissions of the original packet will arrive after the data was scheduled to be rendered, and will be discarded by the application. This gives a lower bound on T_{playout} for standard TCP retransmission to be useful.

The corresponding upper bound is the maximum acceptable delay for the application,T_{max}. If we assume media encoding delay is negligible, $T_{\text{sender}} \approx T_{\text{framing}}$. By combining these bounds, we see that standard TCP retransmissions will arrive in time to be rendered to the application, provided:

$$T_{\text{max}} - T_{\text{framing}} - T_{\text{rtt}}/2 \geq T_{\text{playout}} \geq T_{\text{rtt}} + (3+1) \times T_{\text{framing}} \tag{4}$$

This inequality is shown graphically in Figure 4. The unshaded regions in Figure 4 fall outside of the feasible operating regime of the application and may be ignored, as they correspond to stalls in play-out or overall delay bound violations. The feasible operating regime is represented by the shaded regions that separate useful from wasteful retransmissions. The green cross-hatch highlights the region where standard TCP retransmissions arrive in time to be useful.

Wasteful TCP retransmissions are marked by the red-lined region in Figure 4. When the media play-out delay is less than the retransmission time ($T_{\text{playout}} < T_{\text{rexmit}}$) but satisfies the overall delay bound ($T_{\text{playout}} \leq T_{\text{max}} - T_{\text{framing}} - T_{\text{rtt}}/2$), and is greater than the framing interval ($T_{\text{playout}} \geq T_{\text{framing}}$), then standard TCP retransmissions will arrive too late.This is where inconsistent retransmissions are useful: when a TCP retransmission will arrive too late to replace the original lost packet in this region. By contrast an inconsistent retransmission can use that retransmission slot to transmit the next unsent data segment. The lost packet is never recovered, but its sequence number is reused to send data that will be useful when it arrives.

B. Inconsistent Retransmissions and Real-Time Media

The benefits of TCP Hollywood can be quantified by substituting real-time traffic parameters into Equation 4. Consider interactive voice telephony. Widely deployed speech codecs typically use $T_{\text{framing}} = 20\text{ms}$ with a delay bound of $T_{\text{max}} = 150\text{ms}$ [20]. Assuming media encoding delays are negligible, so that $T_{\text{sender}} = T_{\text{framing}}$, then the feasible region where standard TCP retransmissions arrive in time to be useful can be derived from Equation 4 as:

$$130\text{ms} - T_{\text{rtt}}/2 \geq T_{\text{playout}} \geq T_{\text{rtt}} + 80\text{ms} \tag{5}$$

which has valid solutions for T_{playout} provided $T_{\text{rtt}} \leq 33.33\text{ms}$. This round-trip time bound is low for wide-area networks. For example, TCP retransmission would be useful for calls from the authors' homes within Europe, but discarded during inter-continental calls. Figure 4 shows TCP Hollywood provides valid solutions for T_{playout} when $T_{\text{rtt}} \leq 220\text{ms}$, showing the utility of inconsistent retransmissions for this application.

For on-demand video streaming using the MPEG DASH framework, the framing interval and delay bounds are typically much larger. A typical deployment today might use an encoding segment size of $T_{\text{framing}} = 2\text{s}$, and an overall delay bound of $T_{\text{max}} = 30\text{s}$. Assuming $T_{\text{sender}} = T_{\text{framing}}$, and substituting into Equation 4, this permits valid solutions for T_{playout} provided $T_{\text{rtt}} \leq 13.33\text{s}$, giving no benefit from inconsistent retransmission.

These two applications represent extremes in terms of latency bounds: voice telephony has tight latency bounds, while those of on-demand video streaming are relaxed. We analyse a third application: IPTV delivery using DASH. IPTV applications seek to minimise *zap time* (i.e., the total time taken between a viewer selecting a channel, and content from that channel being displayed). Bouzakaria et al. [21] show that end-to-end latencies – the time between encoding and decoding of a frame – of less than 240ms can be achieved using DASH. Using their techniques, segments are fragmented into 200ms chunks for delivery, giving $T_{\text{sender}} = T_{\text{framing}} = 200\text{ms}$. An overall delay bound of $T_{\text{max}} = 1\text{s}$ allows for channel surfing to be supported. Substituting these values into Equation 4, we see that regular TCP retransmissions do not benefit this application for any RTT values. In contrast, inconsistent retransmissions in TCP Hollywood can be used when $T_{\text{rtt}} \leq 1\text{s}$.

Application	$T_{\max}$ (ms)	RTT Bound (ms)		Useful within a continent?		Useful intercontinental?	
		Standard	Hollywood	Standard	Hollywood	Standard	Hollywood
Voice telephony	150	33.3	220	Y	Y	N	Y
On-demand video	30 000	13 333.3	52 000	Y	Y	Y	Y
Live video	1000	0.0	1200	N	Y	N	Y

Table I. SAMPLE TCP AND TCP HOLLYWOOD RTT BOUNDS REQUIRED TO MEET APPLICATION BOUNDS, HIGHLIGHTING INDICATES WHERE TCP HOLLYWOOD IS BENEFICIAL

Table I summarises the three applications considered. Utility of inconsistent retransmission is seen to depend on the latency bounds of the application. Interactive applications, where the overall latency requirements are tight, can strongly benefit from the ability to send new data in place of a retransmission, but those applications with relaxed latency bounds find less benefit.

C. *Connecting Head-of-Line Blocking*

If a packet is lost, then TCP will send a retransmission once a triple duplicate ACK is received. If standard TCP is used, then later segments will not be delivered to the application until the retransmission of the lost segment is received, potentially causing media play-out to stall. This is known as head-of-line blocking, as discussed in Section II.

The size of the play-out buffer relative to the round-trip time and media framing interval determines whether play-out stalls, or whether there is sufficient buffering to cover the retransmission delay. As was shown in Equation 3, if $T_{\text{playout}} \geq T_{\text{rexmit}} + T_{\text{framing}}$, then the retransmission will arrive in time to be played out, and no head-of-line blocking will occur.

However, if $T_{\text{playout}} < T_{\text{rexmit}} + T_{\text{framing}}$, then the retransmission will not arrive in time to be played-out. This will cause a 1-segment gap in the media play-out, since some data is missing (this occurs with both standard TCP, and with the TCP Hollywood extensions). If standard TCP is used, then the receiver may *also* suffer head-of-line blocking and be unable to access later segments, leading to a longer gap in play-out.

If the retransmission arrives less than one framing interval after it was scheduled to be played out, i.e., if $T_{\text{rexmit}} \leq T_{\text{playout}} < T_{\text{rexmit}} + T_{\text{framing}}$ then it will arrive before the following packet is to be played. In this case, there is no head-of-line blocking, and only a single packet gap occurs in play-out. If it is further delayed, such that $T_{\text{playout}} < T_{\text{rexmit}}$, then head-of-line blocking will cause one or more later frames to also miss their play-out.

A graphical representation is provided by Figure 5. The yellow cross-hatch region in Figure 5a is the region of T_{playout} values where blocked segments will be made wasteful. The details are labeled in Figure 5a by numbered events, with associated time-lines in Figure 5b. For a given value of T_{rtt} the process begins with a loss marked by the red '$\times$'. The next frame arrives at ① and is held by TCP, as are all the segments that follow, awaiting the retransmission. For any size of T_{playout} at that moment ②, the retransmission will arrive too late. Upon arrival of the retransmission ③ TCP releases blocked segments to the play-out buffer. The duration of the head-of-line blocking that will be discarded by the play-out buffer is labelled as T_{HoL} in Figure 5, can be calculated as:

$$T_{\text{HoL}} = T_{\text{rexmit}} - T_{\text{playout}} = T_{\text{rtt}} + 3 \times T_{\text{framing}} - T_{\text{playout}} \qquad (6)$$

The duration translates to N_{HoL} frames missing their play-out due to head of line blocking, and in addition to the retransmission that arrived too late, where:

$$N_{\text{HoL}} = \max\left(\left\lceil \frac{T_{\text{rtt}} + 3 \times T_{\text{framing}} - T_{\text{playout}}}{T_{\text{framing}}} \right\rceil, 0\right) \qquad (7)$$

Finally, we remark on the grey shaded region in Figure 5a that occurs when that when retransmissions arrive past the acceptable deadline. From Equation 4, values of T_{playout} are upper-bound by the application deadline. Subsituting this into Equation 6 gives a lower bound on T_{HoL} of:

$$T_{\text{HoL}} \geq 3 \times T_{\text{rtt}}/2 + 4 \times T_{\text{framing}} - T_{\max} \qquad (8)$$

As T_{rtt} increases under TCP, so too does T_{HoL}, and with it the fragility of the real-time connection. While a TCP retransmission under these circumstances will always arrive too late, the TCP Hollywood extensions eliminate T_{HoL}. In doing so the grey shaded region in Figure 5a, where real-time connections may be infeasible under TCP, are made viable with TCP Hollywood.

Our analysis identifies the value of inconsistent retransmissions, and the way in which they interact with head-of-line blocking. Specifically, it shows that removal of head-of-line blocking, via receiver side modifications to the kernel TCP stack, is necessary to make effective use of inconsistent retransmissions. For this reason, a full deployment of TCP Hollywood eliminates head-of-line blocking, to support latency reduction and improve good-put due to inconsistent retransmissions.

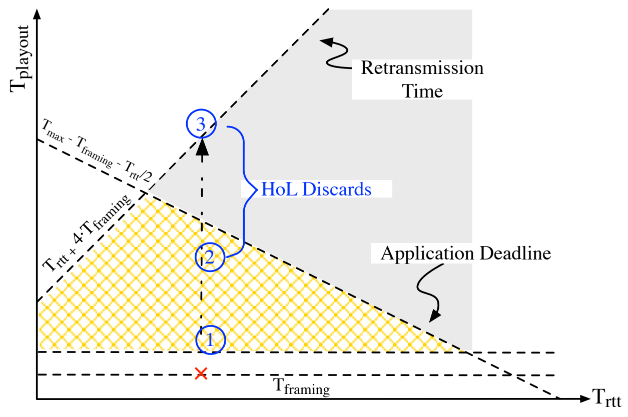
V. IMPLEMENTATION AND DEPLOYMENT

To evaluate our design, we have an implementation of TCP Hollywood that has been tested in fixed and mobile networks in the UK to evaluate ease of deployment.

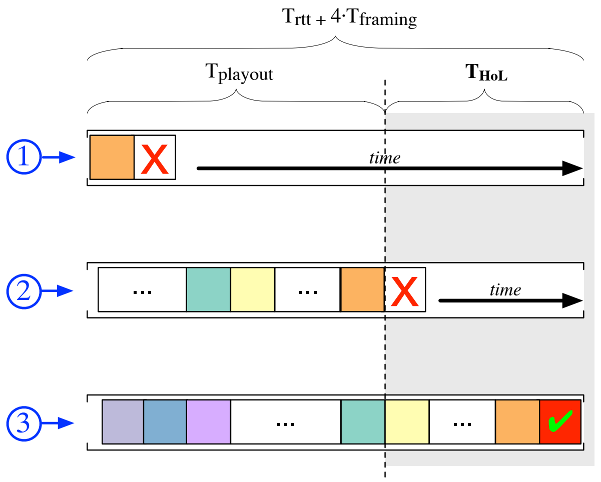
A. *Implementation*

We implemented TCP Hollywood in the FreeBSD 10.1 operating system. The TCP modifications in the kernel impact approximately 300 lines of code, while the intermediary layer comprises 600 lines of user-space C code. The source code is available at http://dx.doi.org/10.5525/gla.researchdata.291.

The main implementation complexity in TCP Hollywood comes from the use of inconsistent retransmissions, since they cause the TCP RTT estimator to interact closely with the message deadlines and dependency tracking features of TCP Hollywood. Figure 6 shows sample results from a *dummynet* testbed used to validate our inconsistent retransmission implementation. These simulate a voice telephony scenario, like that



(a) $T_{playout}$ region where blocked segments will be delivered too late.



(b) Head of line blocking events between a loss and its retransmission.

Figure 5. *Head of line blocking* for real-time applications using regular TCP: for any given RTT and playout, segments that immediately follow a loss (1) are pushed past the acceptable deadline (2), and delivered as late as (3). The gap between RTT and playout is the duration of useful HoL blocked segments that become wasteful

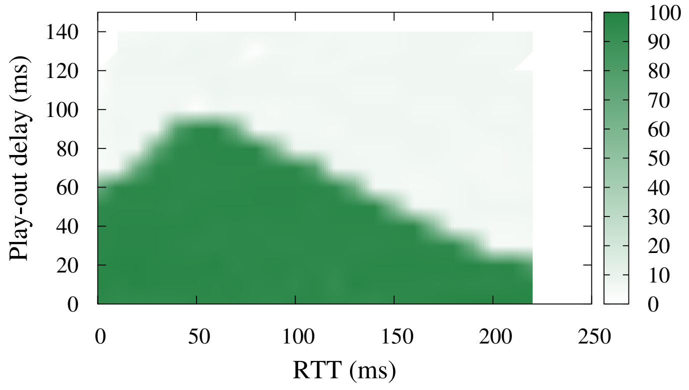


Figure 6. Implementation validation test. Shading shows the percentage of all retransmissions that are inconsistent at different combinations of play-out delay and RTT, for a VoIP scenario. This validates Figure 4

described Section IV-B, using 20ms framing, 120 byte payload per frame, and a maximum one-way delay of 150ms. The colours in Figure 6 show the fraction of retransmissions sent as inconsistent retransmissions, when subject to 5% random packet loss. RTT and playout delay were sampled at 10ms spacing across the entire range, with each point being repeated 5 times. We see that inconsistent retransmissions are triggered as expected, based on the analysis in Section IV. The step-like nature of the curve is due to the 10ms sampling interval. The fuzzy regions around the edge of the coloured space stem from limitations in the timer resolution that cause some degree of unpredictability in whether a retransmission will be inconsistent or not. This test shows that our implementation works as expected, but does not evaluate performance. A detailed evaluation of the implementation performance compared to the analytical results is for future work.

B. *Feasibility of Deployment*

We investigate the feasibility of deploying TCP Hollywood, using results from initial experiments with our FreeBSD implementation on residential and mobile networks in the UK.

TCP Hollywood ought to be entirely compatible with TCP. The only on-the-wire visible difference between a TCP Hollywood flow and a standard TCP flow appears within the payload data carried by inconsistent retransmissions. Recall from Section III-A that inconsistent retransmissions carry new payload data inside of segments with previously transmitted sequence numbers. This modification is invisible to receivers and middleboxes that only process TCP/IP headers, but is visible to middleboxes that use deep packet inspection if they compare the contents of a retransmitted packet with the original data. Depending on the configuration such behaviour may disrupt the connection. For example, a firewall may interpret inconsistent retransmissions as belonging to a man-on-the-side attack, and reset the connection.

We conducted experiments with a live deployment of TCP Hollywood to obtain an initial assessment on whether such middleboxes exist, and what impact they have. A TCP Hollywood server was setup on a public IP address, and configured to always send inconsistent retransmissions in lieu of the original data, so that all retransmissions contained new data with the same sequence numbers. The server was configured to listen on ports 80, 4001, and 5001. Port 80 is used by web traffic, and can be expected to be affected by middleboxes such as “transparent” caches and firewalls. We expect ports 4001 and 5001 to be less likely to be subject to interference by middleboxes, since they are not used by popular applications.

Clients were deployed across a number of access networks, operated by different service providers. Each client connected to the server, and received data. All incoming segments to the client host were recorded by `tcpdump`, then filtered by `iptables` to uniformly drop 5% of segments before reaching the TCP stack for traffic from ports 80 and 4001, leaving traffic from port 5001 unaffected.[2] Each loss induced at the client triggered an inconsistent retransmission from the server. Remaining segments were passed up the stack to the client application, as normal. Data received by the client application was recorded, and compared against `tcpdump` logs from the server to identify the dropped segments, and to compare the payload data in the dropped segments with that sent in the original packet and in the inconsistent retransmission. This allows us to see what segments have been dropped, and to confirm that both the original and retransmission cross the path between client and server, and whether the inconsistent retransmission was delivered.

The evaluation was conducted using clients in 14 different locations in the UK, connecting to a server located at the University of Glasgow. The clients connected via eight different fixed-line residential ISPs (Andrews & Arnold, BT, Demon, EE, Eclipse, Sky, TalkTalk, and Virgin), and four mobile operators (EE, O2, Three, and Vodafone). All of the fixed-line residential ISPs successfully delivered the inconsistent retransmissions. In contrast only one out of the four mobile operators delivered inconsistent retransmissions. The three remaining mobile operators delivered the original segments instead, while the server saw no corresponding segment loss. The observed behaviour is consistent with a transparent split-connection TCP performance enhancing proxy cache that intercepts and responds to ACKs from the client on behalf of the server. On two of the three providers, this caching behaviour was seen on both port 80 and port 4001, while the other provider appeared to operate a cache on port 80 only.

Crucially, TCP Hollywood continued to operate whether or not the provider middlebox was present in the network. At no time did connections suffer a reset, and the use of the TCP Hollywood extensions did not affect connectivity or performance. Middlebox manipulations such as caching are designed to be transparent, leaving the client to believe it is interacting with a standard TCP server. Recall from Section III-C that TCP Hollywood is designed for partial deployment. This experiment provides evidence that TCP Hollywood continues to deliver messages and eliminate head-of-line blocking, even when inconsistent retransmissions are absent. In the worst-case, performance is the same as TCP without our extensions.

The set of networks tested is by no means exhaustive. Further, and larger scale, evaluation is needed to build evidence that inconsistent retransmissions are deployable. Previous studies provide room for optimism, however. Honda et al. [17] investigated deployment of TCP modifications with regards to middlebox interaction, from 142 networks in 24 countries, in early 2011, including inconsistent retransmission measurements taken over a large number of paths, with path diversity. Their observations mirror ours: the majority of paths deliver inconsistent retransmissions as expected, while a small number deliver the original instead. They also observed connection resets on one path, representing less than 1% of paths evaluated.

[2]Given that our goal is to test the ability to deploy TCP Hollywood, rather than performance, we are only concerned with creating sufficient loss to trigger inconsistent retransmissions. A high *un-correlated* drop rate enables TCP to survive where it would fail against correlated drops. The ensuing reduction in throughput translates to reduced loss due to congestion. Thus the client is more likely to see both the original transmission and its retransmission.

VI. Related Work

The immediate precursors of TCP Hollywood are the Minion protocol suite [22] and TL-TCP [23]. The Minion protocol suite includes uTCP, which proves a COBS-encoded user-space datagram abstraction atop TCP, with prioritization and out-of-order delivery. uTCP also provides an API that enables applications to replace existing datagrams in the transmission buffer before they are sent. Datagrams that have already been sent (i.e., those being retransmitted) cannot be replaced. The authors acknowledge this as a conservative design choice, made to ensure middlebox interaction.

Our wire compatibility experiments from Section V-B, and those of Honda et al. [17], indicate that inconsistent retransmissions are possible, but that the integrity of the sequence space needs to be preserved. The need to consider middlebox interaction with new or modified protocols is underscored by the design, and success, of Multi-Path TCP [24]. The design of TCP Hollywood builds on a number of protocols, and tweaks to TCP, that are unlikely to be deployable.

TL-TCP marks the first appearance of time-lines and inconsistent retransmissions [23]. The underlying mechanism works by injecting gaps into the sequence space. This modification is observable by middleboxes, and so is unlikely to be deployable. TCP Hollywood builds on TL-TCP, and related protocols, but does so while focussing on deployability. As discussed in Section III, we minimise changes to the wire protocol to maximise compatibility with middleboxes.

Transport protocols that rely on application-layer metadata to improve performance include Partially Error Controlled Connection (PECC) [25] and PRTP-ECN [26]. Other protocols such as SCTP [7] and DCCP [8] were engineered to broaden the delivery models offered by the transport-layer. Despite standardization and deployment in mainstream operating systems, their use is hampered by a lack of middlebox support.

Liang and Cheriton in [27] note that loss can be more detrimental to streaming application performance than jitter. On-demand streaming applications, for example, can effectively hide jitter from the application but are unable to tolerate loss. The authors present a modified TCP, TCP-RTM, that allows receivers to read beyond a gap in the receive buffer. The sequence numbers in the gap are ACKed, preventing their retransmission by the sender. Applications read from the socket at a predetermined play-out rate offset by some delay. There are no changes to TCP itself; instead, the interaction between application and receiver buffer is modified. Selective negative ACKs (NACKs) allow senders to be informed of the segments that were skipped over.

Deadline-aware TCP is a modified TCP specifically for datacenters, and implements flows with soft time constraints [28]. The modifications allow for the TCP window size and congestion back-off to be varied based on the flow congestion deadline. Flows with imminent deadlines benefit from larger windows. As the network becomes congested, flows will tend to complete closer to their deadlines. The modifications require ECN support in the network, and a modified TCP sender. Requiring ECN support effectively prevents deployment outside of datacenters.

QUIC (Quick UDP Internet Connections) [29] is a transport-layer protocol implemented atop UDP. It incorporates a number of latency-reducing techniques (e.g., large initial data transfers, low RTT setups) that are slowly migrating to TCP. Its use of UDP as a substrate provides an interesting contrast to our choice of TCP. The main motivation being the ability to deploy without kernel modifications, and so QUIC is implemented entirely in userspace. The flexibility of a userspace implementation comes at the cost of universal deployment, since the initial estimates by the QUIC authors show around 5-10% are behind UDP-blocking firewalls. Upon detection of a blocking device QUIC is forced to fall back to TCP. The analysis presented in Section IV shows that falling back to TCP Hollywood is better for latency-sensitive applications (such as those using QUIC), in certain network conditions.

The trade-off between a UDP-based protocol with fall-back to standard TCP, as chosen by the QUIC authors, and a slightly modified TCP variant, as we have chosen, hinges on ease of implementation and deployment. We believe our implementation is simpler, since we build on the TCP infrastructure, but acknowledge that this gives us less flexibility to evolve the protocol. Equally, we believe our implementation is likely to be more deployable, as it builds on TCP. Broader measurement studies, for both TCP Hollywood and QUIC, are needed to evaluate this claim, however.

VII. Conclusions and Future Work

In this paper we presented TCP Hollywood, a modified TCP for real-time multimedia. The analysis shows that our modifications are beneficial to applications with tight latency bounds, such as voice telephony and live video delivery. Further, we've shown that by limiting the wire-visible modifications, we can maintain TCP's widespread ease of deployment.

Future work includes real-world performance evaluation. Measuring the performance improvements, in terms of the increase in usable bytes delivered to the application, that TCP Hollywood provides to the applications analysed in real networks is key to validating the analysis in Section IV. Beyond this, we are exploring enhancements to TCP Hollywood that may further improve performance. For example, dependency information is currently used to determine when *not* to send a message, but it may be a cause *to* send a message, even if that message may not arrive in time to be played out, to allow future messages to be processed. Broader enhancements, such as integration with SACK or MP-TCP, should also be studied.

TCP Hollywood exists within a transport-layer protocol design space that is constrained by ossification. We have TCP and UDP as substrates, with little room for modification. Substrate selection presents trade-offs: TCP gives a wider deployment story than UDP, but depending on the desired functionality, receiver-side kernel modifications can be needed. These trade-offs may shift over time, as the network responds to large deployments of substrate-based transports. For example, QUIC is seeing non-trivial deployment by being included within Google's web browser, and may result in fewer firewalls blocking UDP. This is a long-term concern, however, and in the near future we believe that protocols like TCP Hollywood offer important advantages relating to middlebox traversal, that will make them easy and valuable to deploy. Our initial results show TCP Hollywood is deployable on *all* major fixed and mobile operators in the UK, and offers compelling latency advantages.

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session initiation protocol," IETF, June 2002, RFC 3261.

[2] C. Jennings, T. Hardie, and M. Westerlund, "Real-Time Communications for the Web," *IEEE Communications*, vol. 51, no. 4, Apr. 2013.

[3] M. Cha, P. Rodriguez, J. Crowcroft, S. B. Moon, and X. Amatriain, "Watching television over an IP network," in *Proc. Internet Measurement Conference*. ACM, October 2008.

[4] T. Stockhammer, "Dynamic adaptive streaming over HTTP – standards and design principles," in *Proc. MMSys*. ACM, February 2011.

[5] Cisco, "Visual Networking Index: Forecast and Methodology, 2012-2017," White Paper, May 2013.

[6] D. D. Clark and D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," in *Proc. ACM SIGCOMM*, 1990.

[7] R. Stewart, "SCTP," RFC 4960, IETF, Sep. 2007.

[8] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340, IETF, Mar. 2006.

[9] S. Hätönen *et al.*, "An Experimental Study of Home Gateway Characteristics," in *Proc. Internet Measurement Conference*. ACM, 2010.

[10] C. S. Perkins, M. Westerlund, and J. Ott, "WebRTC: Media transport and use of RTP," IETF, Nov. 2014, work in Progress.

[11] S. Cheshire and M. Baker, "Consistent Overhead Byte Stuffing," in *Proc. ACM SIGCOMM*, 1997.

[12] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, no. 5, May 2012.

[13] N. Khademi, R. Ros, and M. Welzl, "The New AQM Kids on the Block: An Experimental Evaluation of CoDel and PIE," in *Proc. Global Internet Symposium*. Toronto, ON, Canada: IEEE, April 2014.

[14] C. Jin, D. Wei, and S. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," in *Proc. IEEE Infocom*, Mar. 2004.

[15] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," in *Proc. SIGCOMM Conference*. London, UK: ACM, August 1994.

[16] S. McQuistin and C. S. Perkins, "Reinterpreting the Transport Protocol Stack to Embrace Ossification," in *Proc. IAB Workshop on Stack Evolution in a Middlebox Internet*, Zürich, Switzerland, Jan. 2015.

[17] M. Honda *et al.*, "Is it still possible to extend TCP?" in *Proc. ACM IMC*, Berlin, Germany, Nov. 2011.

[18] J.-R. Ohm, "Advances in Scalable Video Coding," *Proc. IEEE*, vol. 93, no. 1, pp. 42–56, Jan 2005.

[19] S. Cheshire and M. Baker, "Consistent Overhead Byte Stuffing," in *Proc. ACM SIGCOMM*, 1997.

[20] ITU-T, "One-way transmission time," Rec. G.114, May 2003.

[21] N. Bouzakaria, C. Concolato, and J. L. Feuvre, "Overhead and performance of low latency live streaming using MPEG-DASH," in *Proc. 5th Intl. Conf. Information, Intelligence, Systems and Applications*. Crete, Greece: IEEE, 2014.

[22] M. F. Nowlan, N. Tiwari, J. Iyengar, S. O. Amin, and B. Ford, "Fitting Square Pegs Through Round Pipes: Unordered Delivery Wire-Compatible with TCP and TLS," in *Proc. USENIX NSDI*, San Jose, CA, Apr. 2012.

[23] B. Mukherjee and T. Brecht, "Time-lined TCP for the TCP-friendly delivery of streaming media," in *Proc. IEEE ICNP*, 2000.

[24] C. Raiciu *et al.*, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *Proc. USENIX NSDI*, vol. 12, 2012.

[25] B. Dempsey, T. Strayer, and A. Weaver, "Adaptive Error Control for Multimedia Data Transfer," in *Proc. IWACA*, vol. 92, 1992.

[26] K.-J. Grinnemo and A. Brunstrom, "Evaluation of the QoS offered by PRTP-ECN - a TCP-compliant partially reliable transport protocol," in *Proc. IWQoS*, Karlsruhe, Germany, Jul. 2001.

[27] S. Liang and D. Cheriton, "TCP-RTM: Using RTP for Real Time Multimedia Applications," May 2002, submission to IEEE International Conference on Network Protocols (ICNP).

[28] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-Aware Datacenter TCP (D2TCP)," in *Proc. ACM SIGCOMM*, 2012.

[29] J. Iyengar and I. Swett, "QUIC: A UDP-based secure and reliable transport for HTTP/2," Work in progress, IETF, Jun. 2015.