TCP Goes to Hollywood

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

Real-time multimedia applications use either TCP or UDP at the transport layer, neither of which offer all of the features required. Ossification means that novel protocols that *do* offer these features are unlikely to be deployed. We present TCP Hollywood, a protocol that is wire-compatible with TCP, while offering an unordered, partially reliable message-oriented transport service. Our analysis shows that this protocol extends the feasibility of using TCP for real-time multimedia applications. Preliminary experiments also show that TCP Hollywood is deployable, with safe failure modes, across all major UK fixed-line and cellular networks.

1. INTRODUCTION

Real-time multimedia applications comprise a large and growing portion of all Internet traffic [1]. The characteristics of these applications – partial reliability based on strict latency bounds, interdependencies between messages – aren't supported by widely deployed transport layer protocols. This is bad: developers reimplement common functionality, applications interact poorly with each other, and the stability of the network is compromised.

A number of efforts to improve this situation, including SCTP [2] and DCCP [3], are hindered by transport layer ossification [4]. Application developers are left to choose between TCP and UDP. Each presents a tradeoff: TCP has wider deployment, but introduces latency, while UDP is better for real-time multimedia applications, but lacks both congestion control and reliability. Our goal is to provide all of the transport service features required by real-time multimedia applications, with the deployment reach of TCP.

Our proposed solution is TCP Hollywood, an unordered, time-lined transport layer protocol that has wire-compatibility with TCP. TCP Hollywood reduces TCP's latency tax, by removing head-of-line blocking and relaxing the reliability guarantee to work within the application's latency bounds. It is message oriented, allowing interdependencies between messages to be expressed.

Our contributions can be summarised as:

- A novel, deployable protocol whose API better supports real-time multimedia applications
- Analysis showing that TCP Hollywood should reduce latency (versus TCP) in realistic network scenarios
- Preliminary evaluations that show that the protocol is deployable, with safe failure modes

Other efforts in this space make design decisions that make deployability unlikely. For example, Time-Lined TCP (TLTCP) [5] leaves holes in TCP's sequence space, making it incompatible with a number of middleboxes [6]. The design of TCP Hollywood focuses on deployability, minimising the extent to which it deviates from TCP's wire protocol, and increasing its compatibility with middleboxes.

We structure the remainder of this paper as follows. Section 2 describes the requirements of a deployable transport-layer protocol for real-time multimedia applications. Section 3 describes the design of TCP Hollywood, and how it fulfills the requirements. Section 4 highlights the combination of network and application parameters where our protocol will help. Section 6 discusses deployability evaluations conducted using our implementation. Section 7 describes related work, while Section 8 concludes.

2. REQUIREMENTS

We adopt the terminology used by the Transport Services (taps) [7] working group at the IETF. A *transport service feature* is an end-to-end feature provided by the transport layer, such as ordered delivery. Our requirements are shaped by two broad goals: to provide the transport service features that are needed by real-time multimedia applications, while maximising deployability.

Our protocol should be message-oriented to allow application data units (ADUs) to be sent; these can be independently processed by the receiver. This allows us to offer out-of-order delivery, removing the latency introduced by enforcing order. Real-time multimedia ADUs have deadlines, by which they must have been played out. If they aren't played out by this time, they are effectively lost. We therefore offer partial reliability: messages will be sent reliably while the sender estimates that they will be useful upon reception, discarding messages that either won't arrive on time, or whose dependencies have not been received successfully. We offer support for multiple substreams, allowing multiple flows (e.g., audio and

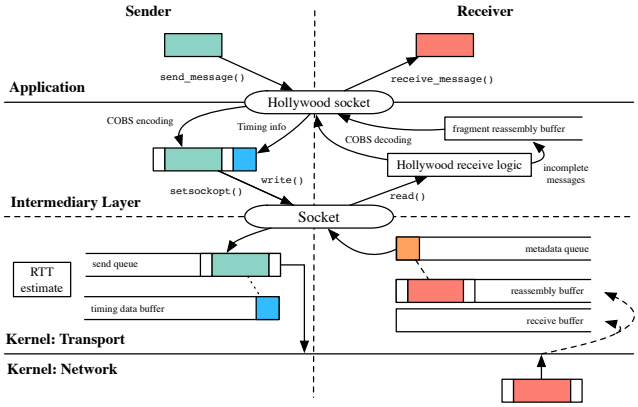

Figure 1: TCP Hollywood architecture ((to do: work in progress..))

video component streams) to be multiplexed across a single transport layer connection. Congestion and flow control are necessary to protect the network and other applications.

Ossification means that protocols that don't look like TCP or UDP on the wire are unlikely to be widely deployable. Therefore, our protocol needs to use either TCP or UDP as a substrate, limiting the wire-visible changes while modifying the end-to-end semantics to implement the features we require.

We select TCP as the substrate for two reasons. The services described in the previous section align better with TCP (e.g., a TCP-friendly congestion control mechanism is already implemented), aiding implementation. Further, TCP has wider deployability than UDP, which is often blocked by enterprise firewalls.

3. DESIGN

In this section, we describe the design of the protocol, showing how it fulfils the requirements discussed in the previous section. The architecture is shown in Figure 1. Broadly, TCP Hollywood is split across an intermediary layer in user-space, and a set of kernel modifications. The intermediary layer works with or without the kernel modifications. This split aids deployability: a minimal set of kernel modifications is easier to implement, and the intermediary layer can always be deployed.

We begin by discussing the sender, going from the application layer down to the network. We then briefly describe the wire protocol, before moving to the receiver, where we work our way back up to the application layer.

3.1 Sender

The sender is responsible for implementing two transport service features: messaging, and partial reliability based on timing and dependency information.

To provide messaging, we encode and frame data at the intermediary layer. Applications pass messages, along with metadata such as timing and dependency information, to the

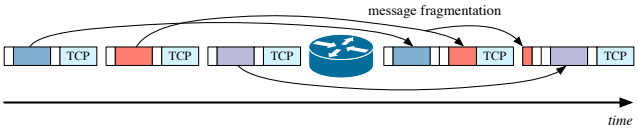

Figure 2: Encoding and framing with leading and trailing markers protects against middlebox re-segmentation; received segments can be properly decoded into messages.

intermediary layer. Our encoding and framing mechanism must enforce message integrity; the receiver must only receive messages that have been sent. As shown in Figure 2, TCP segments can be resegmented and coalesced in the network. We use leading and trailing markers to demarcate messages in TCP's byte stream, requiring an encoding algorithm to remove these markers from the payload. We use consistent overhead byte stuffing (COBS) [8] to remove all zero bytes, using these as framing markers.

Supporting *partial* reliability means that we must relax TCP's existing mechanism. TCP ensures reliability using retransmissions, which introduces both latency and late losses (i.e., segments that are effectively lost because they are too late to be played out). TCP Hollywood has to maintain the retransmission mechanism; middleboxes expect this behaviour, and deployability would be compromised without it. However, the payload of the retransmitted segment may be different under TCP Hollywood. In the kernel, Nagle's algorithm is disabled (i.e., `TCP_NODELAY` is set) to prevent unnecessary buffering.

We use *inconsistent retransmissions* to enable partial reliability. Whenever a message is retransmitted, its associated timing and dependency information is checked to determine if the message will arrive at the receiver in time to be useful. If it will, then the message is retransmitted. Otherwise, the message is replaced by another (i.e., one that passes the timeliness and dependency check) from the queue. This replacement message is sent with the same TCP header as the original. Therefore, a middlebox on the path may observe the same TCP sequence number relating to two or more different payloads. We discuss the implications of this on deployability in later sections.

To maintain message integrity, inconsistent retransmissions will only replace whole messages. Message fragments are not replaced, and are always sent reliably. This ensures that the message reassembly process on the receiver, described in Section 3.3, is correct.

If the kernel modifications are not deployed on the sender, then inconsistent retransmissions are not possible. The implications in terms of performance are analysed in Section 4. The messaging abstraction does not require any kernel modifications, allowing partial deployments (i.e., where only one of the sender or receiver has the kernel modifications deployed).

3.2 Wireline Compatibility

There are many examples of modifications to TCP that see limited deployments because they deviate too far from the wire format of TCP: TL-TCP leaves gaps in the sequence space, for example. It is important to consider middlebox interaction, and to limit wire-visible modifications, if deployability is a concern.

The only wire-visible difference between TCP and TCP Hollywood is the use of inconsistent retransmissions, with everything else – e.g., header format – remaining the same.

3.3 Receiver

The receiver is responsible for implementing the messaging and out-of-order delivery transport service features.

Out-of-order delivery requires modifications to the kernel. Incoming segments are processed as normal; TCP sends the same response as TCP (e.g., a duplicate ACK is sent when an out-of-order segment arrives). However, *all* incoming segments, including out-of-order segments, are queued for delivery to the intermediary layer as soon as they arrive. Therefore, the delivery model for TCP Hollywood differs to that of TCP; segments are delivered to the intermediary layer in *arrival order* rather than *byte order*. Before being passed to the intermediary, segments are tagged with their TCP sequence number to allow message fragments to be reassembled.

Messaging is implemented at the intermediary layer, and operates with or without the out-of-order delivery kernel modifications. Incoming segments contain zero or more complete messages (i.e., data between two zero bytes), which the intermediary layer scans for and queues for delivery to the application. There may be a leading and/or trailing fragment: an incomplete message. The intermediary layer reassembles these fragments using the segment's TCP sequence number. Modifications on the sender ensure that message integrity is maintained (even with inconsistent retransmissions), allowing for reassembly. Once reassembled, messages are queued for delivery to the application.

If the kernel modifications are not deployed on the receiver, the application cannot benefit from the removal of head-of-line blocking. The impact of this is analysed in Section 4.

4. ANALYSIS

A key goal of TCP Hollywood is to reduce transport layer latency. In this section, we quantify the impact that inconsistent retransmissions and the removal of head-of-line blocking has on latency, versus TCP. To do so, we introduce some notation. First, we model the one-way transport delay, T_{owd}, as:

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

T_{sender} is the latency introduced at the sender, resulting from the capture, encoding, and transmission of a media frame. T_{playout} is the application-level receiver-side latency: the sum of the buffering delay, and the decoding and rendering tasks. T_{rtt} is the round-trip time between the sender and receiver. Without loss of generality, the analysis presented here assumes symmetric forward and backward paths.

Next, we model the multimedia itself. T_{framing} denotes the interframe interval of the media, that is, the duration of each frame. We can apply two constraints to this value: (i) $T_{\text{sender}} \geq T_{\text{framing}}$, since a frame must have been fully captured before being sent, and; (ii) $T_{\text{playout}} \geq T_{\text{framing}}$, since we assume that the multimedia is to be decoded (and rendered) without gaps. We approximate that $T_{\text{sender}} \approx T_{\text{playout}}$, given that the encoding time is negligible in comparison to the framing interval. While the decoding time is similarly negligible at the receiver, the de-jitter buffer is likely to be significant, preventing a similar, receiver-side, approximation from being made.

For a given application, there is an acceptable delay bound, T_{max}. Intuitively, $T_{\text{owd}} \leq T_{\text{max}}$, for this bound to be met. Reasonable values for T_{max} vary by application, ranging from the hundreds of milliseconds for voice telephony, to the tens of seconds for on-demand video.

4.1 Inconsistent Retransmissions

To model the impact of inconsistent retransmissions, we quantify the time taken for a retransmission to occur. The receipt of a triple duplicate acknowledgement by a TCP sender results in the retransmission of the lost data. From this, we can model the time taken for a sender to recognise packet loss as:

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

A further T_{framing} compensates for the lost framing interval at the receiver. Therefore, we can put a lower bound on T_{playout} such that retransmitted data arrives on time to be useful:

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

Intuitively, the upper bound is the application's acceptable delay bound, T_{max}. As given above, we approximate $T_{\text{sender}} \approx T_{\text{framing}}$. Combining these upper and lower bounds, we see that TCP retransmissions are useful when T_{playout} is bound as:

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

We show this graphically in Figure 3. The unshaded regions show where the application's delay bounds cannot be met without stalls in play-out. The green hatched shaded region indicates where TCP retransmissions arrive in time to be useful.

The red lined region shows where TCP retransmissions are sent, but do not arrive in time to be useful. It is in this region that inconsistent retransmissions increase network utility: rather than waste bandwidth by retransmitting data that will not be used, TCP Hollywood takes the opportunity to send the next usable message instead. The message that would

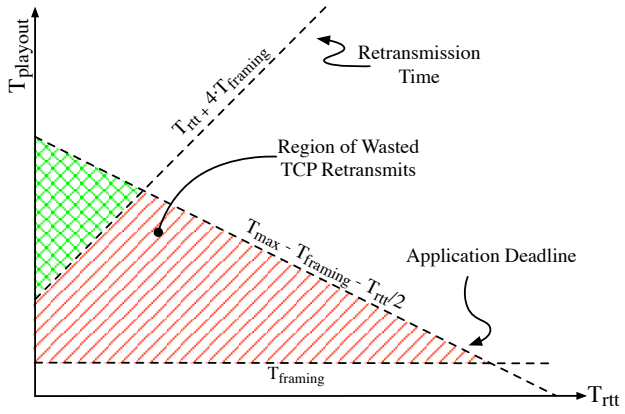

Figure 3: *Inconsistent Retransmissions* for real-time applications: TCP retransmissions may arrive too late for playout delays that must meet the application deadline.

normally have been retransmitted is lost, as it would have been under TCP, but no bandwidth is wasted in resending it.

Inconsistent retransmissions reduce latency and transport layer loss by sending queued messages ahead of the time that they would have been sent under TCP. Both of these performance improvements are beneficial to real-time multimedia applications.

4.2 Head-of-Line Blocking

The interaction between TCP's in-order and reliable delivery service features manifests itself in head-of-line blocking: segments that arrive after a lost segment are not delivered until the lost segment arrives. Intuitively, this can result in underflow of the playout buffer if it is not appropriately sized, causing stalls in media playback. If TCP retransmissions arrive in time to be useful (as determined by the analysis presented in the previous section), then head-of-line blocking will not cause playback stalls – there is sufficient buffering to cover the time taken to retransmit.

However, where the playout delay is not sufficient to cover the time taken to retransmit (i.e., $T_{\text{playout}} < T_{\text{rexmit}} + T_{\text{framing}}$), then the segment is effectively lost, and a one segment gap occurs in the playback. In TCP, head-of-line blocking means that later segments may also effectively be lost whilst they are blocked. The remainder of this analysis determines the cases where head-of-line blocking impacts on media playback.

Head-of-line blocking does not impact media playback when the retransmission arrives less than one framing interval after its play-out time: $T_{\text{rexmit}} \leq T_{\text{playout}} < T_{\text{rexmit}} + T_{\text{framing}}$. However, if the delay is longer than this (such that $T_{\text{playout}} < T_{\text{rexmit}}$), one or more later segments will be effectively lost due to head-of-line blocking. We model the duration of the impact (i.e., the duration of the discarded frames) of head-of-line blocking as T_{hol}:

$$T_{\text{hol}} = T_{\text{rexmit}} - T_{\text{playout}} = T_{\text{rtt}} + 3 \times T_{\text{framing}} - T_{\text{playout}}. \quad (5)$$

From this, the number of discarded frames can be expressed

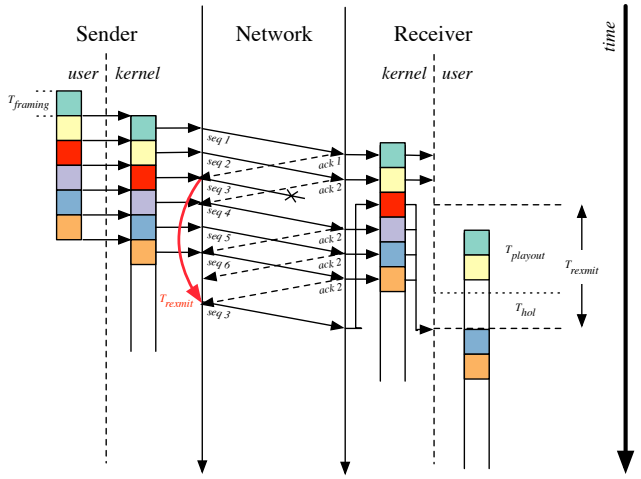

Figure 4: The relationship between T_{playout}, T_{rexmit}, and T_{hol} in TCP

as:

$$N_{\text{hol}} = \max\left(\left\lceil \frac{T_{\text{hol}}}{T_{\text{framing}}} \right\rceil, 0\right). \quad (6)$$

We express this analysis graphically in Figure 4. In this example, segment 3 is lost, and is retransmitted. Segments 4, 5, and 6 are blocked waiting for this retransmission to arrive. As shown, T_{playout} is less than T_{rexmit}, causing the retransmitted segment to arrive too late to be used. Additionally, segment 4 is also effectively lost (despite being at the receiver), as T_{hol} is greater than zero.

Using Figure 4 to summarise the overall impact of TCP Hollywood versus regular TCP, we see two behaviours: (i) inconsistent retransmissions will be triggered for the retransmission of segment 3, increasing network utility, decreasing latency, and improving goodput; and (ii) segment 4 would be played out successfully.

The impact of head-of-line blocking on the application-level loss rate of applications is significant; it amplifies the underlying network loss rate. To ensure that the performance impact of its removal is maximised, messages sent by applications should have some independent utility.

5. MULTIMEDIA APPLICATIONS

The previous section described the combination of network conditions and application parameters within which TCP Hollywood operates. In this section, we apply this analysis to a real-time multimedia application, to show that TCP Hollywood supports these applications in realistic conditions where TCP does not.

We consider an IPTV application that uses DASH for delivery. One of the quality of experience factors of such applications is *zap time*: the total time between a viewer selecting a channel, and content from that channel being displayed. We use techniques described by Bouzakaria et al. [9] to minimise the latency caused by using DASH. Segments are fragmented

into 200ms chunks, giving $T_{\text{sender}} = T_{\text{framing}} = 200\text{ms}$. The overall delay bound, T_{max}, is set to 1 second to allow for channel surfing.

Substituting these application parameters into Equation 4, we see that TCP retransmissions offer no benefit for this application, for any round-trip times. Further, if T_{playout} is less than T_{rexmit}, additional segments (i.e., beyond the retransmitted segment) will be lost. While TCP Hollywood does not prevent these retransmitted TCP segments from being lost, it sends useful data through inconsistent retransmissions. Additionally, head-of-line blocking is removed by TCP Hollywood.

When operating in the red lined region of Figure 3, both TCP and TCP Hollywood will incur some message loss at the application layer. For TCP, this is the number of retransmitted segments multiplied by N_{hol} – for every segment that is retransmitted and arrives too late to be useful, there is a further N_{hol} segments lost due to head-of-line blocking. In TCP Hollywood, the upper bound on application-level loss is the number of retransmitted segments. Removing head-of-line blocking removes its impact on loss. Further, inconsistent retransmissions decrease the loss rate by increasing the amount of useful data sent.

It is clear from this example, and the analysis presented in the previous section, that TCP Hollywood extends the feasibility of using TCP for real-time multimedia applications. Real-time multimedia applications are sensitive to loss, and so reducing the application-level loss rate is important for their stability and the quality of experience provided to users.

6. DEPLOYABILITY

To evaluate the deployability of TCP Hollywood, we conducted evaluations between hosts on residential and cellular networks in the UK, and a server running TCP Hollywood. We implemented the protocol in the FreeBSD 10.1 kernel. The kernel modifications impact around 300 lines of code, while the intermediary layer consists of 600 lines of code.

Inconsistent retransmissions are the only wire-visible difference between TCP and TCP Hollywood; middleboxes on the path may observe to TCP segments with the same sequence number but with different payloads. This behaviour is consistent with man-in-the-side attacks, and so firewalls may disrupt connections that use inconsistent retransmissions.

While such middleboxes may exist, the scope of these preliminary evaluations is to broadly determine the impact they might have, both in terms of the scale of their deployment, and the action they take. A TCP Hollywood server on the public Internet was set to always send inconsistent retransmissions rather than regular retransmissions; all retransmissions contained new data. The server listened on three ports: 80, 4001, and 5001. Port 80 is used to determine if different behaviour is likely for well-known applications (HTTP in this case). For all ports, all segments were logged using `tcpdump`.

Clients were configured to record all incoming segments, and drop 5% of incoming segments on ports 80 and 4001, with no such filter on port 5001. The client indicates that packet loss has occured, and in theory, triggers an inconsistent retransmission. The payloads of segments received are compared to those sent, allowing us to confirm that both the original data and the new data in the inconsistent retransmission cross the path between the client and server.

ISP	Port	
	80	4001
Fixed-line		
Andrews & Arnold	I	I
BT	I	I
Demon	I	I
EE	I	I
Eclipse	I	I
Sky	I	I
TalkTalk	I	I
Virgin Media	I	I
Cellular		
EE	O	O
O2	O	O
Three	I	I
Vodafone	O	I

Table 1: Deployability evaluation results, measuring the delivery of inconsistent retransmissions. I indicates that inconsistent retransmissions were delivered, O indicates that the original data was delivered, and F indicates connection failure. No connection failures were observed.

The ISPs tested, and their results, are shown in Table 1. The behaviour observed for the three cellular networks that do not deliver inconsistent retransmissions is consistent with that of a transparent TCP proxy cache, with the TCP connection between our server and the client being split across the cache. The server does not see the loss, and is not given the opportunity to retransmit. The lost segment is served from the cache.

While clients on those networks that do not deliver inconsistent retransmissions do not benefit from this technique, their performance is no worse than that of TCP. We did not observe any networks where the connection was disrupted. While our evaluations are by no means exhaustive, they can be taken together with those conducted by Honda et al. [**?**], using 142 networks in 24 countries. They observe similar behaviour: most paths deliver inconsistent retransmissions, with some delivering the original instead. Connections are reset in less than 1% of the paths.

7. RELATED WORK

TCP Hollywood builds on Minion [10] and Time-Lined TCP (TL-TCP) [5]. The Minion protocol suite includes uTCP, a COBS-encoded unordered, datagram abstraction built atop

TCP. The prioritisation API allows applications to replace data in the send buffer. In contrast to TCP Hollywood, this replacement can only occur if the data has not yet been sent. TL-TCP introduce timeliness and consistent retransmissions. Gaps are inserted into the sequence space, making deployability unlikely. Honda et al. indicate that the sequence space should be complete. The design and deployability of Multi-Path TCP (MP-TCP) [11] highlights the need to consider middlebox interaction.

There are many other protocols designed for multimedia applications, including Partially Error Controlled Connection (PECC) [12] and PRTP-ECN [13]. In the broader transport protocol design space, SCTP [2] and DCCP [3] aim to widen support for applications. The difference between TCP Hollywood and these protocols is our focus on deployability; our modifications to TCP are constrained to ensure middlebox compatibility.

The design of QUIC (Quick UDP Internet Connections) [14] highlights the trade-off in selection an appropriate transport-layer substrate. QUIC offers a connection-oriented protocol that reduces latency (versus TCP). By building on top of UDP, QUIC can be implemented entirely in userspace, allowing for rapid and widespread deployment. However, the QUIC authors show that deployability is not as wide as if TCP had been used. Our use of TCP constrains the extent to which we can modify the protocol, but we believe that we benefit from wider deployability. Further measurement studies are necessary to show this.

8. CONCLUSIONS

We have presented TCP Hollywood, a transport protocol designed for support interactive multimedia applications. Our analysis shows that the main features of TCP Hollywood – inconsistent retransmissions and unordered delivery – reduce latency (versus TCP), improving performance. We also conducted preliminary deployability evaluations, indicating that widespread deployment is feasible, given the constrained set of modifications that our protocol makes to TCP.

In order to verify that our analysis holds, future work includes real-world performance evaluations. These evaluations will include measuring both the proportion of usable bytes delivered to the application, and the average latency of messages delivered to the application. This mirrors our analysis: we show that inconsistent retransmissions increase utility, while the removal of head-of-line blocking significantly reduces latency. Other future work includes further enhancing the performance of the protocol. At present, the dependency metadata provided by applications could be used as a reason to overrule decisions made based on the timing data. For example, a message may be too late to be played out itself, but it may have many live dependents, and so should be sent.

Ossification has constrained the transport protocol designed space: protocols that do not look like TCP or UDP on the wire are not widely deployable. Protocols such as TCP Hollywood that challenge the design assumptions of these substrates may help to reduce ossification, if deployed at scale. For example, Google's QUIC has significant deployment, and so may see a reduction in the number of firewalls blocking UDP. While UDP remains blocked at its current levels, TCP-based protocols such as TCP Hollywood offer greater deployability. We have shown that TCP Hollywood is deployable on *all* major fixed and cellular operators in the UK, and that it offers non-trivial latency advantages to real-time multimedia applications.

9. REFERENCES

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

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

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

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

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

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

[7] G. Fairhurst, B. Trammell, and M. Kühlewind, "Services provided by IETF transport protocols and congestion control mechanisms," Internet Engineering Task Force, Internet-Draft draft-ietf-taps-transports-09, Jan. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-taps-transports-09

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

[9] N. Bouzakaria, C. Concolato, and J. L. Feuvre, "Overhead and performance of low latency live streaming using MPEG-DASH," in *In Proceedings of the 5th International Conference on Information, Intelligence, Systems and Applications*. IEEE, 2014, pp. 92–97.

[10] 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.

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

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

[13] 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.

[14] J. Iyengar and I. Swett, "QUIC: A UDP-Based Secure

and Reliable Transport for HTTP/2," 00, IETF, Jun. https://tools.ietf.org/html/draft-tsvwg-quic-protocol- 2015.