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, yet neither of these protocols offer all of the features required. Deploying a new protocol that does offer these features is made difficult by ossification: firewalls, and other middleboxes, in the network expect TCP or UDP, and block other types of traffic. We present TCP Hollywood, a protocol that is wire-compatible with TCP, while offering an unordered, partially reliable message-oriented transport service that is well suited to multimedia applications. Analytical results show that TCP Hollywood extends the feasibility of using TCP for real-time multimedia applications, by reducing latency and increasing utility. Preliminary evaluations also show that TCP Hollywood is deployable on the public Internet, with safe failure modes. Measurements across all major UK fixed-line and cellular networks validate the possibility of deployment.

CCS Concepts

•**Networks** → **Network protocol design; Transport protocols;**

Keywords

Transport protocols; real-time multimedia applications

1. INTRODUCTION

Real-time multimedia applications comprise a large and growing portion of all Internet traffic [4]. The characteristics of these applications, namely their tight latency bounds and interdependencies between messages, are unsupported at the transport layer. Consequently, developers are forced to reimplement common functionality, applications interact poorly with each other, and the stability of the network is compromised [15] [1].

Our goal is to develop a transport protocol that provides all of the features required by real-time multimedia applications, without compromising deployability. Previous efforts suggest that an entirely new protocol is unlikely to see wide deployment and use [10], as evidenced by the deployment of protocols such as SCTP [23] and DCCP [14]. Therefore, we must build the services we require on top of either TCP or UDP. We select TCP as the substrate for our protocol, given the complexity of implementing TCP-friendly congestion and flow control atop UDP, and the wider deployment of TCP in enterprise networks.

We present TCP Hollywood: an unordered, time-lined, transport layer protocol, that supports partial deployment. Critically, TCP Hollywood is wire-compatible with TCP and so is feasible to deploy on the Internet. In TCP Hollywood, TCP's latency tax is eliminated by (i) removing head-of-line blocking, and (ii) relaxing TCP reliability guarantees to respect application latency bounds. In addition, TCP Hollywood uses a message-oriented abstraction so that interdependencies between messages can be expressed.

Previous efforts in this space have made important contributions toward broadening transport layer APIs and reducing latency for real-time applications. Time-Lined TCP (TLTCP) [17], for example, allows applications to express timelines for messages, with data only transmitted during its timeline. In Minion [18], applications can replace unsent messages in the sending buffer. Our work builds on these efforts, with a focus on deployability. TCP Hollywood eliminates deviations from TCP's wire-visible protocol, making it resilient to middlebox interference and modification.

This paper offers three main contributions. First, a novel and deployable protocol with an API that better supports real-time multimedia applications. Second, preliminary analysis that reveals when TCP latency exceeds application bounds, and when TCP Hollywood improves utility. Finally, we report on preliminary evaluations that show that TCP Hollywood is deployable, with safe failure modes.

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 reveals the combination of network and application parameters where our protocol will help, and Section 5 extends this analysis to a multimedia application. Section 6 discusses deployability evaluations conducted using our implementation. Section 7 describes related work, while Section 8 concludes.

2. REQUIREMENTS

We outline the requirements we address with TCP Hollywood, using 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 features that are needed by real-time multimedia applications, and to ensure that these are deployable on the wider Internet.

Applications require a message-oriented protocol, to allow independently decodeable application data units (ADUs) to be sent. This abstraction also enables out-of-order delivery, removing the latency introduced by enforcing order. Real-time multimedia ADUs have an

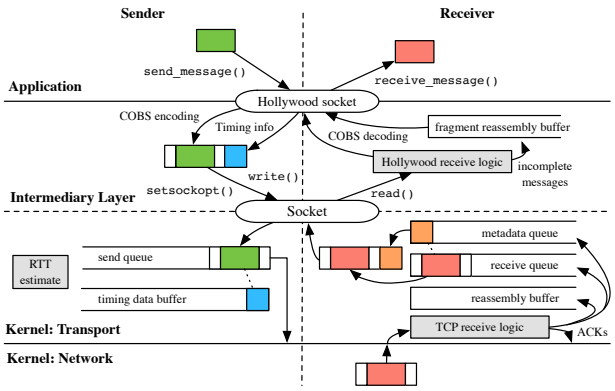

Figure 1: TCP Hollywood architecture

associated deadline by which they must be played out, after which their data is essentially lost. Further, messages may be dependent on other messages that have not been received. This suggests a need for partial reliability: messages will be (re-)transmitted when they are estimated to be useful upon reception, and an alternative sent otherwise. Multimedia applications also benefit from the ability to multiplex multiple streams across a transport-layer connection, allowing them to separate audio and video streams. This may benefit applications using scalable video encoding (for example, using the H.264/SVC codec) [19], where different layers could occupy different sub-streams. Finally, fairness, and congestion and flow control are necessary to protect the network and other applications.

Deployment is constrained by Internet ossification [9]. Protocols that fail to present as TCP or UDP on the wire are unlikely to be forwarded by middleboxes [10]. Therefore our protocol must appear to be TCP or UDP on the wire, while modifying the end-to-end semantics to implement the required features. We select TCP as the substrate for two reasons. The services we describe align better with TCP (e.g., TCP already provides congestion control), reducing implementation complexity. Further, TCP has wider deployment than UDP, which is sometimes blocked by enterprise firewalls.

3. DESIGN

In this section, we describe the design of TCP Hollywood, showing how it fulfils the requirements discussed in Section 2. The architecture is shown in Figure 1. Broadly, TCP Hollywood is split across an intermediary layer in userspace (providing the API, and messaging abstraction), and a set of kernel extensions (supporting *inconsistent* retransmissions and out-of-order delivery).

Our design presentation follows messages from the sender, going from the application layer down to the network. We then briefly describe the wire protocol, before discussing the receiver-side as messages are delivered to the application. A more detailed description of the design can be found in [16].

3.1 Sender

The sender implements two features of TCP Hollywood: (i) messaging, in the intermediary layer, and (ii) partial reliability based on timing and dependency information, in the kernel.

To provide messaging, we encode and frame data at the intermediary layer. Applications first pass messages, and optional metadata such as timing and dependency information, to the intermediary layer. The intermediary layer prepends the sub-stream identifier to the message data, and this is then encoded and framed before being

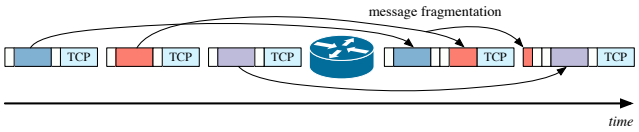

Figure 2: Encoding and framing with leading and trailing markers protects against middlebox re-segmentation; received segments can be properly decoded into messages.

passed to the transport layer. Figure 2 shows the mapping between messages in TCP Hollywood and segments in TCP. To maximise utility and reduce latency, TCP Hollywood *sends* each message in a single segment. Segments can be re-segmented and coalesced in the network; a message may be split across multiple segments, or one segment may contain multiple messages. Leading and trailing markers are used to retain the message boundaries. In our design, consistent overhead byte stuffing (COBS) [3] escapes all zero bytes, for use as framing markers. This process is efficient, expanding the payload by at most 0.4%, and transparent to the application: the content that can be sent is not affected.

In the kernel, Nagle's algorithm is disabled (i.e., `TCP_NODELAY` is set) to prevent buffering. Nagle's algorithm buffers small application writes, amortising the size of the TCP header across a larger segment. The benefit of this is not significant where application messages are large relative to TCP headers, such as in our target applications, and is incompatible with our goal of reducing latency.

Supporting *partial* reliability means relaxing 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 useful) – it takes time for the sender to become aware of the loss, and for the retransmitted segment to arrive at the receiver. TCP Hollywood conforms to the syntax of TCP's retransmission mechanism so as to be compatible with middleboxes and maintain deployability. However, TCP Hollywood may send *inconsistent* retransmissions, that is, the payload of the retransmitted segment may differ from the original.

Whenever a message is retransmitted, the sender uses timing and dependency information to estimate if the message will be useful to the receiver. If so, the message is retransmitted. Otherwise, the message is replaced by another from the queue that *will* be useful. This replacement message is sent with the same TCP header as the original. 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, only whole messages will be replaced in inconsistent retransmissions. Where the retransmitted TCP segment contains only part of a message (i.e., a message fragment), this will not be replaced. If the sender replaced fragments then the reassembly process on the receiver (described in Section 3.3) may deliver a message to the application that is a composite of fragments from different messages.

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 on either sender or receiver, maximising deployability.

The computational overhead at the sender consists of the encoding process at the intermediary layer, and the maintenance of per-message metadata in the kernel. COBS encoding is not computationally expensive [3], but is currently performed by the intermediary layer, and so requires taking a copy of the message. Tighter integra-

tion with applications, such that the COBS encoding is performed as part of the multimedia encoding, would remove this copy. In the kernel, metadata (e.g., timeline and dependency data) is stored about each message, until the message can no longer be sent.

3.2 Wireline Compatibility with TCP

Deviating too far from the wire format of TCP compromises deployability. Middleboxes in the network perform functions that make assumptions about the behaviour of TCP hosts, and reset connections when unexpected behaviour is observed. As an example, TLTCP [17] creates holes in the TCP sequence space. When a middlebox observes an incomplete sequence space, it assumes that the sender is misbehaving, and closes the connection. It is important to consider middlebox interaction, and to limit wire-visible modifications, if deployability is a priority.

The only wire-visible difference between TCP and TCP Hollywood is the use of inconsistent retransmissions, where the payload of a retransmitted segment differs from that of the original (the checksum is recalculated, and padding used to ensure that the size remains the same). Everything else (i.e., the syntax and semantics of the TCP header) remains unchanged. Therefore, TCP Hollywood is only visible to middleboxes that perform payload inspection. Broadly, there are two reasons for a middlebox to perform this inspection: to enhance performance through caching, and to improve security by detecting anomalies. Split-connection TCP caches are widely deployed where round-trip times are high and non-congestive packet loss is common; we observe that the majority of UK mobile providers cache segments in this way. These caches deliver the original segment rather than the retransmission, removing the performance benefit of TCP Hollywood, but without disrupting the connection. Firewalls that perform deep packet inspection (DPI) are designed to detect anomalies, including protocols that behave unexpectedly. These firewalls are slow and computationally expensive at scale, and so are generally limited to enterprise networks where the trade-off is worthwhile; we did not observe middleboxes of this type during our deployability evaluations. A firewall that detects inconsistent retransmissions would likely reset the connection. Disabling inconsistent retransmissions upon detecting this behaviour is future work. We consider deployability further in Section 6, with experimental results from an initial deployment.

3.3 Receiver

The receiver implements two features of TCP Hollywood: (i) messaging, at the intermediary layer, and (ii) out-of-order delivery, in the kernel.

Out-of-order delivery requires modifications to the kernel. All incoming segments generate a metadata entry, with their TCP sequence number and length, which is added to a queue. A copy of the payload is stored when the segment is out-of-order, or when there are segments in the reassembly buffer. TCP Hollywood and TCP generate the same response to out-of-order segments. When the intermediary layer reads from the socket, it receives the next segment in the metadata queue, tagged with its TCP sequence number to allow for reassembly. This is a different delivery model to TCP: segments are delivered in *arrival* rather than *byte* order.

Messaging is implemented at the intermediary layer, operating with or without the kernel modifications. Incoming segments contain zero or more complete messages (i.e., data between two zero bytes), which are queued for delivery to the application. There may be a leading and/or trailing message fragment, which are reassembled using the segment's TCP sequence number. Once reassembled, messages are decoded, before being 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.

The computational overhead at the receiver consists of a decoding process, again performed at the intermediary layer, and so requiring a copy of the data. Beyond maintaining the metadata structures described earlier, the kernel makes an additional copy (vs. TCP) of the payload for segments that arrive out-of-order, or while there are segments in the reassembly queue. This additional copy could be removed by optimising our proof-of-concept implementation.

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:

$$T_{\text{owd}} = T_{\text{sender}} + T_{\text{playout}} + T_{\text{rtt}}/2 \quad (1)$$

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 delay.

Next, we model the multimedia itself. T_{framing} denotes the duration of the application data within each frame (i.e., the inter-frame interval). 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 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 low hundreds of milliseconds for VoIP, to the tens of seconds for on-demand video.

4.1 Inconsistent Retransmissions

TCP Hollywood's inconsistent retransmission mechanism reduces wasted bandwidth, sending a usable message where regular TCP would retransmit data that cannot be played out. To model the impact, 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, the time taken for a sender to recognise packet loss is:

$$T_{\text{rexmit}} = T_{\text{rtt}} + 3 \times T_{\text{framing}} \quad (2)$$

A further T_{framing} is needed to account for the corresponding framing interval that is lost at the receiver. The lower bound on T_{playout} if retransmitted data is to arrive on time to be useful becomes:

$$T_{\text{playout}} \geq T_{\text{rexmit}} + T_{\text{framing}} \quad (3)$$

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:

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

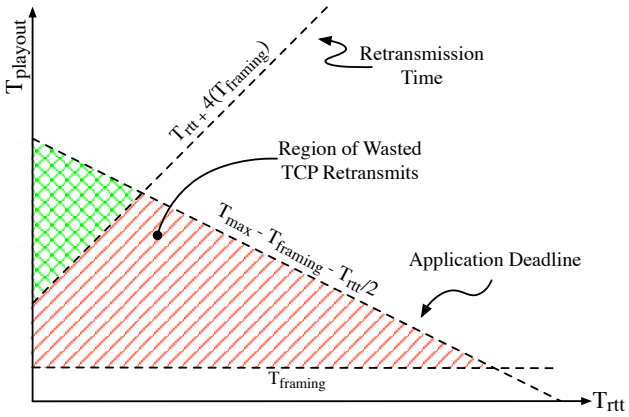

Figure 3: *Inconsistent retransmissions* operate within the red lined region where TCP retransmissions would arrive too late

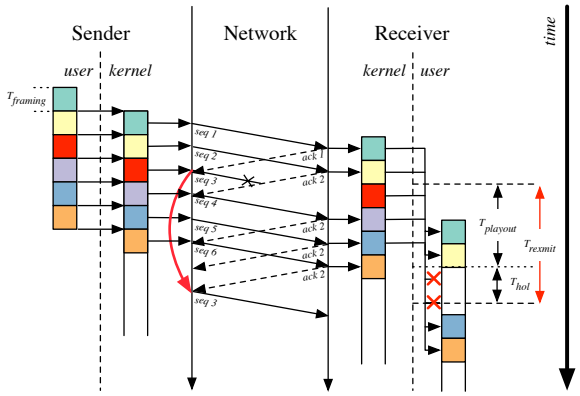

Figure 4: The relationship between T_{playout}, T_{rexmit}, and T_{hol} in standard TCP

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. Inconsistent retransmissions operate in this region: rather than resending data that won't arrive in time to be played out, TCP Hollywood sends the next usable message instead. The original message is still lost, but no bandwidth is wasted by resending it when TCP Hollywood is used. Inconsistent retransmissions reduce latency and 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 Section 4.1), then head-of-line blocking will not cause playback stalls – there is sufficient buffering to cover the time taken to retransmit.

When the playout delay is less than the time taken to retransmit (i.e., $T_{\text{playout}} < T_{\text{rexmit}} + T_{\text{framing}}$), the segment is effectively lost, and a one segment gap occurs in the playback. In standard TCP, head-of-line blocking is also invoked. If the retransmission arrives less than one framing interval after its playout time, head-of-line blocking has no impact on media playback, as $T_{\text{rexmit}} \leq T_{\text{playout}} < T_{\text{rexmit}} + T_{\text{framing}}$.

However, say the retransmission *fails* to arrive at least one framing interval before its playout time. Then at least one of the later, head-of-line blocked, segments will be discarded by the application. The duration of the impact (i.e., the duration of the discarded frames) can by modelled 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 as:

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

A graphical representation of head-of-line blocking in standard TCP is provided in Figure 4. In this example, segment 3 is lost, and must be retransmitted. When this retransmission arrives, the segment is delivered to the application, along with the blocked segments 4, 5, and 6. As shown, T_{playout} is less than T_{rexmit}, and so segment 3 has arrived too late to be used. Additionally, given that T_{hol} is greater than zero, head-of-line blocking renders segment 4 useless, despite its on-time arrival.

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 network loss rate. To ensure that the impact of its removal is maximised, messages sent by applications should independently decodeable.

5. APPLICATION TO MULTIMEDIA

Section 4 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 MPEG-DASH [22] for delivery. One of the important 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. Figure 5 plots the region where inconsistent retransmissions are useful, for various framing intervals, as derived from Equation 4. At 60fps, each frame is 16.7ms. T_{framing}, the duration of each message, is determined by the number of frames in the message. We set T_{max} to 1 second, within the recommended zap time [12].

Sending a small number of frames per message allows TCP retransmissions to work, reducing both application-level loss and the average one-way delay of each frame. However, the cost of sending a small number of frames is an increase in overheads: each message must be independently decodeable, reducing the compression efficiency of the codec. In addition, the header overhead is high.

We can reduce these overheads by increasing the number of frames in each message. This allows the multimedia codec to use predictive frames for efficiency, and increases the ratio of payload to headers. This cost of this is shown in Figure 5 – sending more

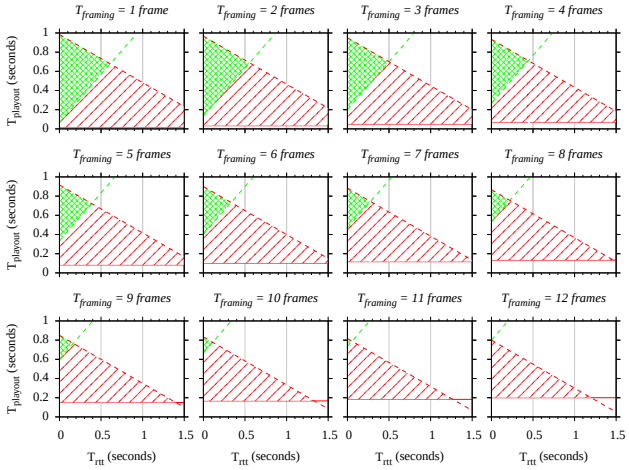

Figure 5: The trade-off between $T_{framing}$ and the utility of TCP retransmissions, which operate in the green hatched region

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.

than 10 frames in a message renders TCP retransmissions useless: they arrive too late to be played out. To recover this lost utility, a protocol such as TCP Hollywood is needed.

TCP and TCP Hollywood incur different rates of application-level loss when operating in the red lined regions of Figure 5. For TCP, this is the number of retransmitted segments (lost because they arrive too late), multiplied by N_{hol}, the number of segments lost due to head-of-line blocking. While TCP Hollywood does not recover the late losses of TCP, inconsistent retransmissions allow applications to send usable data (that would be sent later) instead. Further, head-of-line blocking is removed in TCP Hollywood, eliminating its impact on loss, where T_{playout} is less than T_{rexmit}.

Underlying this example, and the TCP Hollywood protocol, is an important change in the transport service provided to applications. DASH applications are layered on top of TCP, and therefore expect reliability. Under TCP Hollywood, the delivery model is closer to RTP [21] applications running over UDP [5]: the latency added by maintaining order and reliability is removed. This shift is necessary to support low-latency applications, and the techniques proposed by Bouzakaria et al. [2] show that DASH can be used in this way.

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. The source code is available at http://dx.doi.org/10.5525/gla.researchdata.291.

Inconsistent retransmissions are the only wire-visible change in TCP Hollywood; middleboxes may observe segments with the same sequence number but different payloads. Firewalls might interpret this as an attack (e.g., man-on-the-side) and disrupt connections.

Our preliminary evaluations are designed to determine the impact that middleboxes might have on TCP Hollywood. We measure both the scale of their deployment, and any action that they take toward TCP Hollywood connections. A TCP Hollywood server on the public Internet was configured 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 recorded all incoming segments. 5% of those received on ports 80 and 4001 were intentionally discarded to trigger a retransmission, with no synthetic loss on port 5001. The payloads of segments received were compared to those sent, allowing us to confirm that both the original segment and the inconsistent retransmission crossed the path between the client and server.

The ISPs tested, and the results, are shown in Table 1. For the three cellular networks that did not deliver inconsistent retransmission, we observed behaviour consistent with transparent split-connection proxy caching. Our server did not see the loss, and was not given the opportunity to retransmit. The lost segment was instead served from the cache.

Where inconsistent retransmissions are not delivered, TCP Hollywood offers no benefit, but is no worse than TCP. We did not observe any networks where the connection was disrupted. While our evaluations are not exhaustive, they can be taken together with those conducted by Honda et al. [11], across 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 [18] and TLTCP [17]. The Minion suite includes uTCP, a COBS-encoded unordered, datagram abstraction built atop TCP. Its 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. TLTCP introduces timelines, but creates gaps in the sequence space, making deployment unlikely – Honda et al. indicate that the sequence space should be complete. The design and deployability of MultiPath TCP (MPTCP) [20] highlights the need to consider middlebox interaction.

Many other transport protocols have been designed, including SCTP [23] and DCCP [14], that offer novel delivery models, and Partially Error Controlled Connection (PECC) [6] and PRTP-ECN [8], that improve support for multimedia applications in particular.

The difference between TCP Hollywood and these protocols is our focus on deployability and middlebox compatibility.

TCP Hollywood uses inconsistent retransmissions and a message-oriented abstraction to reduce latency, and better utilise the network. This could also be achieved by providing the same message-oriented abstraction to applications and multiplexing messages across multiple paths using MPTCP. Adopting an MPTCP-based strategy alone removes partial deployability, such as can be achieved by TCP Hollywood. Finding ways of combining these methods to maximise both deployability and performance is future work.

Datacenter applications (e.g., Web search) often consist of multiple flows with soft real-time constraints. Deadline-Aware Datacenter TCP (D2TCP) [24] is a modified TCP that allows applications to express deadlines for their flows, which are used to vary the window size and congestion backoff. Flows are given larger windows as they approach their deadlines, ensuring their timely completion. D2TCP requires ECN in the network, and a modified sender. This enables deployment in datacenters, but not in the public Internet.

The design of QUIC (Quick UDP Internet Connections) [13] highlights the trade-off in selecting an appropriate transport-layer substrate. QUIC offers a connection-oriented protocol that reduces latency (vs. TCP). By using 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 modifications we can make, but we believe that we benefit from wider deployability. Further measurement studies are needed to show this.

8. CONCLUSIONS

We have presented TCP Hollywood, a transport protocol designed to support interactive multimedia applications. Our analysis shows that the main features of TCP Hollywood – inconsistent retransmissions and unordered delivery – reduce latency (vs. 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 reduces latency. Other future work includes further enhancing the performance of the protocol. For example, dependency metadata could be used as a reason to overrule decisions made based on timing data. A message may be too late to be played out, but may still be needed for its dependents.

Ossification has constrained the transport protocol design 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] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What happens when HTTP adaptive streaming players compete for bandwidth? In *Proc. NOSSDAV*. ACM, 2012.

[2] N. Bouzakaria, C. Concolato, and J. L. Feuvre. Overhead and performance of low latency live streaming using MPEG-DASH. In *Proc. IISA*. IEEE, 2014.

[3] S. Cheshire and M. Baker. Consistent Overhead Byte Stuffing. In *Proc. ACM SIGCOMM*, 1997.

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

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

[6] B. Dempsey, T. Strayer, and A. Weaver. Adaptive Error Control for Multimedia Data Transfer. In *Proc. IWACA*, volume 92, 1992.

[7] G. Fairhurst, B. Trammell, and M. Kühlewind. Services provided by IETF transport protocols and congestion control mechanisms. IETF, Jan. 2016. Work in Progress.

[8] K. Grinnemo and A. Brunstrom. Evaluation of the QoS offered by PRTP-ECN: A TCP-compliant partially reliable transport protocol. In *Proc. IWQoS*, July 2001.

[9] M. Handley. Why the Internet Only Just Works. *BT Technology Journal*, 24(3), July 2006.

[10] S. Hätönen et al. An Experimental Study of Home Gateway Characteristics. In *Proc. IMC*. ACM, 2010.

[11] M. Honda et al. Is it still possible to extend TCP? In *Proc. ACM IMC*, Nov. 2011.

[12] ITU-T FG IPTV. Consideration on Channel Zapping Time in IPTV Performance Monitoring. Contribution 545, Apr. 2005.

[13] J. Iyengar and I. Swett. QUIC: A UDP-based secure and reliable transport for HTTP/2. IETF, June 2015. Work in Progress.

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

[15] A. Mansy, M. Fayed, and M. Ammar. Network-layer fairness for adaptive video streams. In *Proc. IFIP Networking*. IEEE, 2015.

[16] S. McQuistin, C. Perkins, and M. Fayed. TCP Hollywood: An Unordered, Time-Lined, TCP for Networked Multimedia Applications. In *Proc. IFIP Networking*. IEEE, 2016.

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

[18] 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*, Apr. 2012.

[19] J.-R. Ohm. Advances in Scalable Video Coding. *Proc. of the IEEE*, 93(1):42–56, Jan 2005.

[20] C. Raiciu et al. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proc. USENIX NSDI*, volume 12, 2012.

[21] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[22] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, 18(4), April 2011.

[23] R. Stewart. SCTP. RFC 4960, Sept. 2007.

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