Rethinking ACKs at the Transport Layer

Ana Custura
University of Aberdeen

Tom Jones
University of Aberdeen

Gorry Fairhurst
University of Aberdeen

Abstract—**Acknowledgements are a core component of transport protocols. They can send control information and be used to receive feedback about transmission progress, to measure responsiveness and capacity of the network path, and numerous other purposes. This paper takes a long look at the way acknowledgements are used across different transports and what ACK information is actually needed in new protocol designs. TCP's use of ACKs has inspired the design of other Internet transports and was used as a model for QUIC.**

Index Terms—**ACK, transport protocols**

I. INTRODUCTION

The design of any Internet transport protocol needs to operate safely over a wide range of path characteristics and be robust to changes in the set of devices forming the network path. Most transports rely on feedback from the remote endpoint to inform the local endpoint about the success of transmission across the path. Acknowledgments (ACKs) are often sent as separate packets. They can have any size, but are typically much smaller than data packets. Some transports therefore allow the ACK information to be coalesced with other information sent by a remote endpoint. Many transports send data in one direction and ACKs in the opposite direction.

Many transports allow ACKs to be observed in the network. Some devices intentionally reduce the volume/rate of ACKs to improve performance for asymmetric paths [9]. This can lead to network device ossification when connectivity becomes reliant on seeing ACKs. In response, protocols, such as QUIC [14] have been designed to encrypt and authenticate their ACKs. This prevents in-network modification, and motivates our examination of how this will impact performance.

The remainder of this paper explores how ACKs are used by Internet transport protocols. It examines the usage of ACKs in QUIC and suggests the present design will face performance limits when used over asymmetric paths. It concludes with recommendations for the design of an appropriate ACK policy for future transports such as QUIC.

II. FUNCTIONS OF ACKNOWLEDGMENTS

Although the most familiar use-case of ACKs is to perform retransmission or repair, the feedback of ACKs provides a range of important transport functions [8].

ACKs are used to complete setup of a new flow by determining whether the remote endpoint accepts the communication. This ensures both endpoints associate packets with the flow and understand how to process packets. Many transports support negotiation, allowing endpoints to agree (acknowledge) a set of features and parameters to configure the transport.

ACK Use-case	TCP	SCTP	DCCP	RTCP	QUIC
Connection establishment	x	x	x		x
Param. Reneg.		x	x		x
Path estimation	x	x	x	x	x
Loss detection	x	x	x	x	x
ACK Vector			x	x	x
CC	x	x	x	?	x
ACK-clocking	x	x			
Pacing	?	?	x	x	x

TABLE I: A summary of ACK functions by transport protocol.

This coordinates protocol state at endpoints (e.g., TCP SYN options; SCTP INIT; DCCP feature negotiation).

ACKs have a role in estimating path characteristics. After an endpoint has started communicating, ACKs can be used to adjust the Round Trip Time (RTT) to track potential change in path characteristics [8].

Most reliable transports trigger loss recovery after ACKs fail to confirm delivery by reception. Loss could be detected by observing the time-ordering of received ACKs (as in TCP DupACK, RFC 5681) or by utitilising a timer [8]. Care is needed to avoid interpreting ACK loss (e.g. under persistent congestion) or ACK reordering, as as a signal of data loss.

Internet flows need to implement safeguards to avoid inappropriate impact on other flows that share resources along a path. Reception of ACKs indicate data has successfully traversed the network. Congestion control (CC) algorithms can estimate a safe transmission rate from the rate of reception of ACKs, or the volume of data acknowledged per RTT. In general, CC benefits from frequent feedback of ACKs.

To promote more fair sharing of capacity, senders can limit bursts in transmission (e.g. SCTP, RFC 4960). ACKs can help to pace the forward transmission. This could be explicit (e.g. each ACK releases new data) or implicit (i.e., ACKs drive a rate limiter or burst-mitigation method). This needs at least multiple updates per RTT, but can benefit from additional understanding the timing of data received.

III. ACK POLICIES OF COMMON TRANSPORT PROTOCOLS

This section examines how ACKs are used in different transports. Table I enumerates the protocols examined and identifies relevant ACK functions.

A. Transmission Control Protocol (TCP)

TCP is the most popular transport protocol and currently carries the majority of web and streaming video traffic. It uses a sliding window to provide reliable data reception using a cumulative ACK. This has influenced the design of later protocols. In TCP, ACK packets are used to perform all functions detailed in Section II.

We define the ACK Ratio (AR) as the number of packets received to the number of ACKs sent. Early versions of TCP sent an ACK for each received data packet, resulting in AR 1:1. While this provides ample information, it contributes 50% of packets sent. A TCP receiver can delay sending an ACK for up to two times the Maximum Segment Size of data [2] (AR 1:2). RFC 5681 requires Delayed ACKs to be sent within 500 ms of the arrival of a packet. However, many current receivers use a smaller delay, e.g. 200ms or 40ms.

A TCP CC establishing the path capacity grows the congestion control (CC) window *cwnd* [2] exponentially during slow-start, where each received ACK increases the *cwnd* and the sending rate is controlled by the ACK rate. Delaying ACKs can reduce the rate of *cwnd* growth.

Since Appropriate Byte Counting (RFC 3465), CC operates on the cumulative acknowledged bytes, not on individual ACKs. Delayed ACKs increase the time to reach the capacity where the delay is significant compared to the RTT. For this reason, Delayed ACKs After Slow Start (DAASS) [1] can revert to an AR of 1:1 during slow start. However, a TCP receiver does not in general know which CC is used, so cannot know when slow start has finished. There is therefore no standard algorithm for implementing DAASS. The Linux kernel implements a form of DAASS (TCP QUICKACK) using AR 1:1 for at least the first 8 packets[1].

ACK Delay is suspended during TCP loss recovery. Out of sequence packets are reported using the selective acknowledgement (SACK) option (RFC 2018). An ACK carries up to 3 SACK blocks, each describing gaps in the received window. SACK improves efficiency after loss, but increases the volume of ACKs while a receiver is awaiting retransmission.

B. The Stream Control Transport Protocol (SCTP)

SCTP (RFC 4960) offers connection-oriented multi-streaming, and supports a number of modes, from reliable streams like TCP, to partial reliability. Although based on datagrams, methods resemble TCP. ACKs confirm connection establishment and shutdown, and are used to provide reliability. The SCTP initialisation (and INIT ACK) can be larger than for TCP. SCTP follows TCP guidelines on delayed ACKs, generating an ACK for at least every 2nd received packet. ACK Delay is 200ms.

C. Datagram Congestion Control Protocol (DCCP)

DCCP (RFC 4340) is a datagram transport that supports a set of CC identifiers (CCIDs) [17]. Designed after TCP, it supports an ACK Vector option that can explicitly report

[1]https://linux.die.net/man/7/tcp

individual received packets. Like TCP, new connections start with AR 1:2. A DCCP sender can adjust transport parameters during a connection, using DCCP feature negotiation [17]. This allows a sender to change the AR.

CCID2 (RFC 4341) defines an approximately TCP-friendly method that can adapt the AR, proving it does not exceed cwnd/2 and must be $>= 2$ for a *cwnd* of 4 or more packets. For each *cwnd* of data with 1 lost or marked ACK, the AR is doubled; and is decremented for each *cwnd* $/(AR^2 - AR)$ consecutive *cwnd*s of data with no ACK loss or marking. This does not adapt to path delays, such as queues building; and is unable to account for the cost of link ACK transmission. An AR more than 1:2 would increase burst sizes, unless the sender performs pacing (RFC 4341).

D. The Real Time Protocol (RTP)

RTP (RFC 3550) is a connection-less transport for real time media that is often sent at a target rate. RTP Control Protocol (RTCP) provides control and signals statistics (e.g., loss rate and delay variance) and can be used to provide loss recovery/repair and CC. Reports can be encrypted. RTCP reports are often regularly sent, e.g. a randomised default of five seconds. To reduce ACK frequency, a single packet can carry multiple RTCP messages. The RTCP extended report (XR) packet (RFC 3611) supports an ACK vector for less frequent feedback (e.g., once per RTT, or each 50-200ms).

E. QUIC

QUIC is a new connection-oriented protocol being specified by the IETF [14]. Packets have monotonically increasing numbers to eliminate ACK ambiguity, and provides more precise RTT estimation [15]. Successful transmission is tracked by feedback of cumulative ACK frames. In contrast to TCP and DCCP, QUIC encrypts all header fields including ACKs.

IETF QUIC [14] currently recommends sending an ACK for each alternate ACK-eliciting packet and implementing ACK delay. This mimics the recommended policy for TCP (AR 1:2). The default delay is 25 milliseconds (the value used by emerging implementations such as Chromium and Quicly). A QUIC receiver shares the ACK delay with the sender to improve the precision of RTT calculations. QUIC employs a CC similar to TCP, but adapted to QUIC [12]. Early specifications for QUIC allowed continuing to sending an ACK Frame for every subsequently received packet for up to 1/8 of an RTT after reordering. This significantly increased the volume of ACK information, and resulted in an increase in return traffic volume (especially significant for an asymmetric path) and this is no longer encouraged.

QUIC has been reported to generally outperform TCP with HTTP/2 [15], nevertheless, performance issues can arise over paths that present different characteristics [18] [15]. We use an experimental testbed described in Section IV, to compare QUIC ACKs to other transport protocols. Section V compares the results to the expected overhead.

Bytes/Packet	TCP	SCTP	DCCP	RTCP XR	QUIC
Min/No Loss	20	24	18	<<22*	26
Max/No Loss	26	24	22	22	36
Min/Loss**	52	52	39	30	53
Min/Loss***	84	Unl	Unl.	Unl.	Unl.

* The minimum RTCP ACK is dependent on data rate.
** Min Loss ACK size corresponds to isolated loss of a single packet.
*** Max Loss ACK size corresponds to a pattern of loss that would cause enough received gaps to grow the ACK vector size to a full packet.

TABLE II: Estimated ACK bytes per data packet

IV. EXPERIMENTAL TESTBED METHODOLOGY

Network performance was evaluated using a Linux client and server, and a FreeBSD router to emulate an asymmetric path (e.g. a satellite path [18]). On these paths, the ACKs generated by forward data impact the capacity of the bottleneck return link, and can constrain throughput.

We consider a best case scenario with 8.5Mbps/1.5Mbps forward and return rates and an emulated delay of 600ms [18]. The return capacity depends on the weather conditions, but importantly also on the subscriber traffic sharing a capacity pool. This return traffic includes requests for data, but also social media and video conferencing traffic [22] (e.g., an HD video call using Skype is 1.5Mbps). We therefore also consider a 100kbps scenario, representing the return path capacity available for ACKs at a busy time of day.

When required, traffic shaping emulated a 1% forward path packet loss. We examined the performance of three implementations of QUIC: Quicly, draft revision 27, Chromium, draft revision 26, and PicoQUIC, draft revision 26. Experiments transferred 10MB of forward data from server to client. Network traces and logs were collected and stored for analysis.

The source code of Quicly and Chromium QUIC was modified to enable AR 1:10 in addition to the default AR 1:2. The AR 1:10 for QUIC was chosen as a safe maximum in line with IW. This AR was also recently shown by Fastly to provide improve computational efficiency[2].

V. TRANSPORT PROTOCOL OVERHEAD

The number of ACK bytes per data packet using TCP, SCTP, DCCP, RTP and QUIC was estimated from the protocol specifications. Table II summarises the computed size assuming an Ethernet maximum packet of 1500 bytes.

A TCP connection handshake is normally less than 60B. It can include data using Fast Open (RFC 7413), resulting in a packet up to the MSS (e.g. a 1500B packet). A simple TCP ACK is 40B, and the overhead for a delayed ACK is therefore 40B/2*MSS. More frequent ACKs from DAASS can increase this to 40B/MSS. ACK size increases to 12B when the Timestamp Option (RFC 7323) is enabled. The ACK size after loss further increases, by including SACK ranges (up to

[2]https://www.fastly.com/blog/measuring-quic-vs-tcp-computational-efficiency

20B of options until loss recovery completes). It is possible for TCP to dynamically adjust the AR [19], e.g. ACK-CC (RFC 5690) adjusts the AR to avoid return path congestion. However, we are however not aware of wide-scale deployment.

SCTP connection packets are typically smaller than data packets. A simple SCTP ACK comprises a common packet header (12B) and a SACK chunk (16B with no loss), resulting in 28B. The minimum overhead for a delayed ACK is 28B/2*MSS. A SACK chunk consists of 0 to 65535 gap ACK blocks, which can grow to fill a SCTP packet.

DCCP connection packets are typically smaller than data packets. Each half-connection sends ACKs either as packets or bundled with data packets. Each ACK comprises a Generic Header (minimum 12B) plus ACK Number Subheader (4B) plus IP headers. An AR 1:2 results in 16B/2*MSS (with no reported loss). Loss/reordering can be encoded in an ACK Vector option, covering up to 16192 packets using run-length compression. This can grow to fill a DCCP packet.

The default RTP reporting interval results in an approximate AR 1:2. However, the interval can be adaptive. Multimedia traffic often does not require frequent feedback [20], for stable and responsive CC. A simple RTCP RR packet for a single source is 32B. An RTCP (XR) packet consists of report blocks with a bitmap of lost and received packets. Even with compression, the RTCP packet size can consume capacity out of proportion with other ACK packets.

QUIC connection handshake packets are the same size as the maximum data packet size [14]. All QUIC packets consist of a transport header followed by frames that carry control information and data streams. A simple ACK (a QUIC packet with just an ACK frame) is 51B, assuming 16B for an encryption cypher and an ACK Frame under 10B, resulting in 1.2% overhead (including headers). A variable-length encoding is used for non-negative integer values, with smaller values needing fewer bytes [14], but causing the size of a cumulative ACK to grow as more bytes are sent. A transfer that exceeds 16,384 packets, adds 2B to each ACK and any reported ranges. Following loss [12], QUIC ACKs carry a set of ACK range vectors. The number of bytes to encode an ACK range could grow to fill an entire QUIC packet, unless a smaller limit is configured.

While the size of TCP control packets (ACKs) is limited by the TCP option space, QUIC packets carry a variety of types of frames to coordinate connection state. ACK size therefore varies more than for TCP. Figure 1 presents the distribution of packet sizes (after connection setup) observed on a return path during a 10MB forward transfer. This shows 3 different QUIC implementations, compared to a TCP receiver. Differences in size result from decisions around how frequently to send other control frames. The median values are 52B for TCP, 63B for Chromium, 68B for Quicly and 83B for PicoQUIC.

Table III presents a comparison in the number of bytes sent on the return path by two implementations of QUIC using AR 1:2 and AR 1:10 and Linux TCP using AR 1:2. The total volume of ACK data is observed to be larger than the ACK volume for TCP for the same AR. Table III and Figure 2b also show the impact of a 1% link loss on the

No Loss			
Bytes	Chromium	Quicly	TCP
Sent	10.7 MB	10.7 MB	10.7 MB
Return (1:2)	313 KB	343 KB	242 KB
	3.1%	3.4%	2.4%
Return (1:10)	76 KB	77 KB	121 KB
	0.7%	0.7%	1.2%

1% Loss			
Bytes	Chromium	Quicly	TCP
Sent	10.7 MB	10.7 MB	10.7 MB
Return (1:2)	336 KB	390KB	299 KB
	3.3%	3.9%	2.9%
Return (1:10)	262KB	166KB	150 KB
	2.6%	1.6%	1.5%

TABLE III: Measured volume for two QUIC implementations using AR 1:2 and 1:10, and TCP using AR 1:2, with no link loss and 1% loss. The projected volume of ACK bytes for TCP assumes simple ACK-Thinning that removes every other cumulative ACK (AR 1:4).
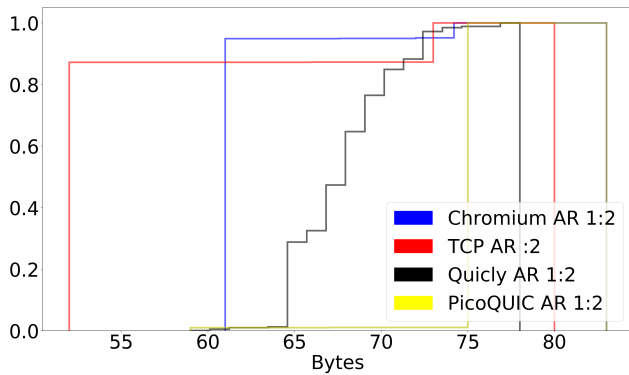

Fig. 1: CDF of return path packet sizes measured after the first RTT, for a 10MB transfer, with no link loss.

forward path. The small increase in volume due to SACK/ACK Range information is evident for both QUIC and TCP. Figure 3 presents the number of ACK packets transmitted per second during a 10MB forward transfer.

VI. ACK MODIFICATION

Although TCP specifies an AR 1:2. Stretch ACKs (a cumulative ACK for more 2 TCP segments) have been observed [1] and are are now common [7] for various reasons.

As flows increase their rate, there is typically an increase in the ACK rate. However, many return paths are asymmetric [9]. That is, the capacity in the available return capacity is much less than that of the forward direction, or the return path *rate* is constrained by the return link technology. Forward data can then generate more ACKs than can be carried over the bottleneck return link, and a queue builds. A reduction in available capacity (reduced to 100kbps) resulted in the forward rate reducing from about 800 to about 400 packets/second using AR 1:2 (Figure 5), corresponding to a drop from
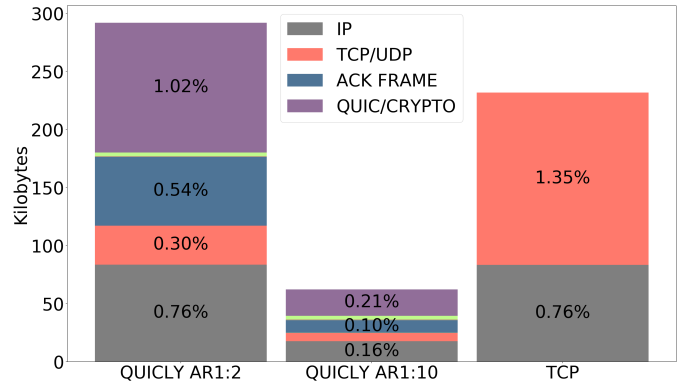
(a) Breakdown of measured return path overhead for QUIC and TCP during a 10MB transfer with no link loss, AR 1:2 and 1:10
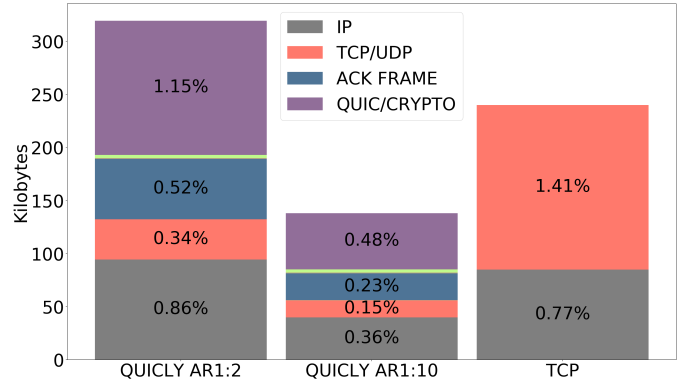
(b) Breakdown of measured return path overhead for QUIC and TCP during a 10MB transfer with 1% emulated forward path link loss, AR 1:2 and 1:10

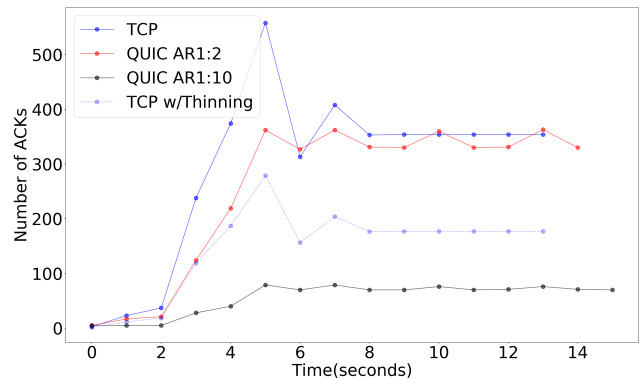Fig. 2: Breakdown of return path overhead for QUIC and TCP

Fig. 3: Number of return packets/sec for TCP and Quicly using AR 1:2 and 1:10, measured each second.

8.5Mbps to 5Mbps. If the same flow uses AR 1:10, it can take full advantage of the 8.5Mbps capacity.

Some routers schedule data and ACKs differently from FIFO, which changes the ordering and/or timing of ACKs. Some other routers schedule based on packet size, although this is discouraged [9]. Although these methods could be used with encrypted packets, they would do so without observing the ACK header, based only on the size of packets, and are hence prone to mistakes.
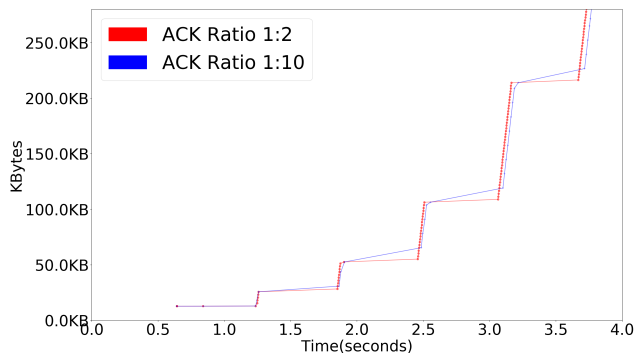
Fig. 4: Measured *cwnd* over time using Quicly with AR1:2 and AR1:10, showing similar CC behaviour.
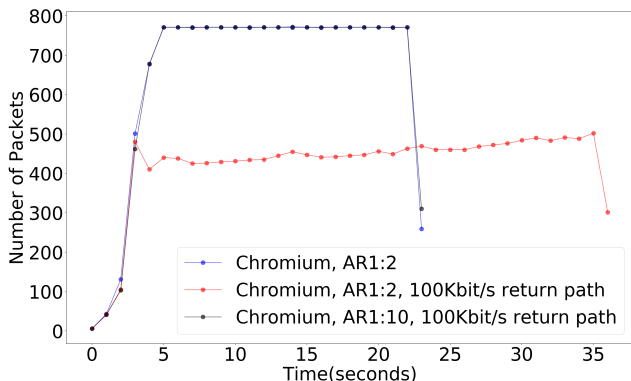

Fig. 5: Measured number of data packets/sec using Chromium with AR1:2 and AR1:10, for a return path that is unconstrained and a path limited to 100kbps of capacity, measured each second.

A. ACK Filtering in the Network

For paths where the return capacity is limited or return link transmission is "expensive", TCP ACK Filtering [9] has been used to "Thin", redundant TCP ACKs [3]. A simple method queues TCP ACKs for the same flow and removes all except except the last cumulative ACK. Even a small reduction (e.g. a factor of 2) can significantly reduce pressure on return path capacity, and reduce queuing delay, benefiting any traffic sharing the bottleneck. These techniques are implemented for various links (e.g., DOCSIS [4], LTE, and WiFi [16]).

B. Performance Enhancing Proxies (PEPs)

A PEP reduces the ACK rate by using a proxy to split the end-to-end transport into a series of network segments. A specialised link transport protocol can then be used for a specific network segment (e.g. radio link), resulting in a much smaller AR. Use of a PEP relies upon observing both data packets and ACKs. Hence, PEPs can not operate with encrypted traffic (e.g., with an encrypted tunnel or using an encrypted transport, such as QUIC).

C. Endpoint Stretch ACKs

ACK processing can consume as much as 20% of CPU cycles in server applications [6]. Many high rate network cards therefore use Large Receive Offload (LRO), to reduce per-packet receive processing. Hardware-agnostic approaches also provide similar functions, e.g., Generic Receiver Offload. These methods also result in Stretch ACKs [11] and corresponding optimisations have been made to TCP to mitigate this by including pacing [21], adapting CC algorithms [5] [23], or using timer-based retransmission.

Although a reduced AR can have benefit, some applications (e.g., low-rate interactive applications or memory-constrained senders) can be negatively impacted by delayed or Stretch ACKs. A sender-controlled TCP mechanism to request an immediate ACK has therefore been suggested [10].

VII. Choosing an Appropriate QUIC ACK Policy

Recognising that QUIC can not benefit from ACK Thinning or PEPs, we present a proposal to change the default QUIC ACK Policy. This seeks to allow operation over asymmetric paths without compromising performance. We propose a receiver default that sends an ACK for at least every 10 packets, while maintaining a minimum number of ACKs per RTT to ensure frequent sender updates.

If slow start is a part of the CC method, performance may benefit when an ACK Frame is sent for at least every 2nd ACK-eliciting packet during slow start (AR 1:2), especially for a small path RTT. This ensures Stretch ACKs do not significantly impact the initial rate of *cwnd* growth. This mimics DAASS in TCP and could, for instance be used for the first 100 received packets.

After slow start, an ACK would be sent when a period more than MIN(max_ack_delay,min_rtt/4) has passed since receiving the oldest unacknowledged data or it has accumulated 10 unacknowledged packets.

Figure 2a shows how AR 1:10 scales the volume of each component of the ACKs by a factor of 5 when there is no link loss. Figure 2b shows that AR 1:10 can reduce the volume of ACKs even with 1% forward path loss. The increased ACK size for Quicly is attributed to ACK ranges sent after loss. Similarly, forward path loss with TCP results in ACKs with SACK information, increasing the size of the TCP header.

QUIC sends an immediate ACK after reordering with ACK range information, increasing the volume of ACK information. Research continues to examine whether removing this will have a significant impact on performance.

Our choice to use an AR 1:10 is balanced by the existing need for QUIC to pace traffic, which is already required for bursts of the order of 10 packets (QUIC's initial window). The overhead (Figure 1) for AR 1:10 with QUIC is comparable to that for TCP with ACK Filtering since a QUIC ACK is 1.5-2 times larger than a TCP ACK (see Table II). In this calculation we assume that ACK Thinning would typically result in a 2-3 times reduction in the TCP ACK rate (Figure 3).

Figure 4 presents the *cwnd* growth. There are notable steps in the plot, at the interval of the 600ms path RTT [18]. These

steps are expected to disperse if there were cross-traffic sharing the bottleneck, or if pacing had been used by Quicly. These results highlight the similarity in sender behaviour with an AR 1:2 and 1:10, confirming that reducing the ACK rate has not negatively impacted the CC in this case. This result was also confirmed with Chromium.

A. Discussion

Sending fewer ACKs or a smaller volume of ACKs, does not necessarily mean less feedback information at the sender. Adapting the AR beyond 1:2 may benefit from a change in the information returned in the ACK. CCs (e.g., BBR) that use pacing, not ACK-clocking, can require only a few (e.g., 4, 8) ACKs per RTT for CC. By reducing the number of network and transport layer headers, there are opportunities to provide more detailed feedback: the ACK Delay in QUIC; ECN reception information; detailed loss reports; etc. This encourages a fresh approach to considering the role of ACKs.

At first glance, it may appear attractive to use a higher AR (e.g. 1:100). This would have benefit at higher transmission rates (where it can reduce the processing cost at endpoints) and for paths with greater asymmetry or cases where protocols can detect congestion on the return path. However, a higher value could potentially negatively impact CC, loss recovery, etc. In many cases, an additional 25% extra delay is unlikely to be an issue, a sender can still only react after feedback has been received This motivates our suggestion to ensure ACK feedback at least each 1/4 RTT.

A higher AR raises additional questions and must consider the many roles an ACK can play within a protocol. New methods could provide richer feedback, increasing detail about packet arrival times in ACKs to help configure pacing, or for a path capacity probe. More detailed timing can help estimate the minimum RTT, jitter, etc and help characterise forward path burst-tolerance, reordering, duplication, etc.

QUIC provides methods that can reconfigure a receiver by sending an update frame for transport parameters [14]. This could change the ACK Policy during a connection [13], or based on CC (as with DCCP CCID-2), but details remain an area for future experimentation. Changes could also be based on the application (as suggested for TCP [10]). Our proposal to change the default AR does not conflict with these methods.

Large service operators could also use this with databases of path characteristics for each client to optimise their ACK policy. However, this could result in significant differences between services, with some having no incentive to compensate for the characteristics of a particular path, risking low levels of actual deployment.

VIII. CONCLUSION

ACKs are used in most transport protocols, but different protocols use different policies, one of which is the choice of an appropriate ACK Ratio (AR). The ACK policy needs to balance the need for effectively growing the *cwnd* at the start of a connection with the desire to efficiently use the path. This must consider a range of use-cases: from reducing the per-packet processing at high-rates within data centres, to reducing overhead for asymmetric paths.

This paper has discussed the benefits for TCP of deploying ACK Thinning to compensate for asymmetry in specific network segments. It then reviewed the current method for QUIC and observed that QUIC this suffers performance penalties when used over asymmetric paths because of the larger volume of ACKs and the inability to rely on in-network ACK filtering deployed for TCP. In response, we propose and evaluate a change to the QUIC transport specification that would use a less conservative default ACK Ratio, reducing ACK volume while retaining or improving performance.

REFERENCES

[1] M. Allman. On the generation and use of TCP acknowledgments. *ACM SIGCOMM CCR*, 28(5):4–21, 1998.

[2] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681, Sept. 2009.

[3] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance. *IEEE/ACM Transactions on Networking*, 5:756 – 769, 01 1998.

[4] Cable Television Laboratories, Inc. *DOCSIS 3.1 MAC and Upper Layer Protocols Interface Specification*, 10 2019.

[5] N. Cardwell. Linux Kernel: Merge branch: fix stretch ACK bugs in TCP CUBIC and Reno, 2015.

[6] M. Chan et al. Improving Server Application Performance via Pure TCP ACK Receive Optimization. In *USENIX Annual Technical Conference*, pages 359–364, 2013.

[7] H. Ding and M. Rabinovich. TCP stretch acknowledgements and timestamps: findings and implications for passive RTT measurement. *ACM SIGCOMM CCR*, 45(3):20–27, 2015.

[8] L. Eggert, G. Fairhurst, and G. Shepherd. UDP Usage Guidelines. RFC 8085, Mar. 2017.

[9] H. B. et al. TCP Performance Implications of Network Path Asymmetry. RFC 3449, Dec. 2002.

[10] C. Gomez and J. Crowcroft. TCP ACK Pull, May 2020. draft-gomez-tcpm-ack-pull-01. IETF Work in Progress.

[11] V. INC. Performance best practices for VMware vSphere 6.7, 2018.

[12] J. Iyengar and I. Swett. QUIC Loss Detection and Congestion Control, July 2019. IETF Work in Progress.

[13] J. Iyengar and I. Swett. Sender Control of Acknowledgement Delays in QUIC, 2020. IETF Work in Progress.

[14] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport, Jan. 2017. IETF Work in Progress.

[15] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove. Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols. In *IMC 2017*, pages 290–303, 2017.

[16] H. Kim, H. Lee, and S. Shin. On the cross-layer impact of TCP ACK thinning on IEEE 802.11 wireless MAC dynamics. *IEICE Transactions*, 90-B:412–416, 02 2007.

[17] E. Kohler, M. Handley, and S. Floyd. Designing dccp: Congestion control without reliability. *ACM SIGCOMM CCR*, 36(4):27–38, 2006.

[18] N. Kuhn, G. Fairhurst, J. Border, and S. Emile. QUIC for SATCOM, 2020. IETF Work in Progress.

[19] S. Landström. *TCP/IP technology for modern network environments*. PhD thesis, Luleå University of Technology, 2008.

[20] C. Perkins. RTP Control Protocol (RTCP) Feedback for Congestion Control in Interactive Multimedia Conferences, Nov. 2019. IETF Work in Progress.

[21] V. Tran and O. Bonaventure. Beyond socket options: making the Linux TCP stack truly extensible. In *IFIP Networking Conference*. IEEE, 2019.

[22] M. Trevisan, D. Giordano, I. Drago, M. M. Munafò, and M. Mellia. Five years at the edge: Watching internet from the isp network. *IEEE/ACM Transactions on Networking*, 28(2):561–574, 2020.

[23] P. Yang. Linux Kernel: Merge branch: fix stretch ACK bugs in congestion control modules, 2020.