Impact of Acknowledgements using IETF QUIC on Satellite Performance

Ana Custura
University of Aberdeen

Tom Jones
University of Aberdeen

Gorry Fairhurst
University of Aberdeen

***Abstract*—This paper explores the performance of IETF QUIC using a GEO satellite service. It first examines how TCP acknowledgements are adapted on paths with high asymmetry and then discusses the implications on the way ACKs are used by a QUIC connection. It then presents experimental results for satellite scenario using two QUIC implementations with different ACK policies. The results show that reducing the number of ACKs sent per packet can allow QUIC to achieve acceptable performance, while also reducing the impact on the satellite return link subsystem. We finally present our results and discuss the benefits of our proposed change.**

***Index Terms*—QUIC, asymmetry, satellite**

I. Introduction

Two main characteristics shape the characteristics of an Internet path using a Geostationary Earth Orbit (GEO) satellite with a transport protocol: path delay and path asymmetry. Both of these impact performance of transport protocols.

Transmission Control Protocol (TCP) is the most commonly used transport protocol in the Internet today. It offers a bidirectional service, using Acknowledgements (ACKs) to confirm which packets have been received by the remote endpoint and to receive feedback about connection progress. ACKs are used to estimate the path round trip time (RTT) and inform congestion control mechanisms. A transport has to use an appropriate ACK rate. Receivers need to send enough ACKs to avoid stalls in flow control or loss of ACKs, where too too infrequently ACKs can limit throughput, although too high an ACK rate can consumes excessive transmission resource [16].

Like many access technologies, an Internet path that includes a satellite access network can exhibit path asymmetry [9]. Capacity provisioned in the forward direction (gateway to a terminal) is often significantly larger than in the return direction (terminal to gateway). If the return path becomes filled with TCP acknowledgements, this can limit the rate that can be achieved on the forward path.

Capacity is not the only asymmetric path constraint. In many Bandwidth-on-Demand (BoD) broadband satellite systems, transmission on the return path also consumes resource and has cost to the user/operator of a satellite segment (controlled by the Radio Resource Management - RRM [10]). Even when the forward throughput is not constrained, consumption of return link resources can impact other applications that share the link or are allocated capacity from the same resource pool [10] [23].

Most current satellite systems use split-TCP Performance Enhancing Proxies (PEPs) [4]. These operate at the transport layer and rely upon being able to observe TCP headers to mitigate the effects of delay and asymmetry. PEPs historically improved TCP performance and also application performance acceleration for HTTP, the main protocol for the web, which uses TCP as its underlying transport. In combination with HTTP/2, the TLS protocol provides application layer security by encrypting the otherwise plaintext HTTP payloads. With the emergence of HTTP/2 and TLS, the main contribution of PEPs changed to providing transport acceleration. This remains effective, because the design of HTTP/2 already brings benefits for a satellite user [5] [22].

A successor to HTTP/2 is emerging in HTTP/3. This uses a new transport, QUIC, to replace TCP - see Figure 1. Google QUIC (gQUIC) [15] was announced in 2012, and was designed to improve user experience for web applications by reducing transport and connection overhead.

A recent activity within the IETF has built upon gQUIC to define the IETF QUIC transport protocol. This is to be standardised in a series of RFCs [13] [11] [25]. QUIC [13] that promise improvements on TCP. The protocol is still under active development, with the working group starting the last call standards process in June 2020. It is important to analyse IETF QUIC, since rapid deployment can be expected soon after publication of the specification, resulting in a fast rise of encrypted traffic.

QUIC provides encryption and authentication for all packets it sends, including ACKs. The security features of QUIC prevent use of split-transport PEPs. This paper aims to understand whether QUIC is disadvantaged in satellite networks compared to other transport protocols. To make a fair assessment of QUIC performance we chose two mature implementations that tracked the emerging standards, and had the required tooling to run our satellite experiments: Chromium QUIC and Quicly.

The paper starts with an analysis of the need and benefit of using a PEP with TCP, presenting baseline measurement data for QUIC, TCP and TCP in the absence of a PEP. It then examines the impact of using QUIC, to determine whether the advances in IETF QUIC benefit users of a satellite system, and whether this compensates for the inability to use a split-transport PEP. In doing so, we evaluate the traffic to consider the impact on the satellite return link. Finally, the paper proposes a change to the QUIC protocol that is expected to benefit broadband satellite systems and other networks with appreciable path asymmetry.

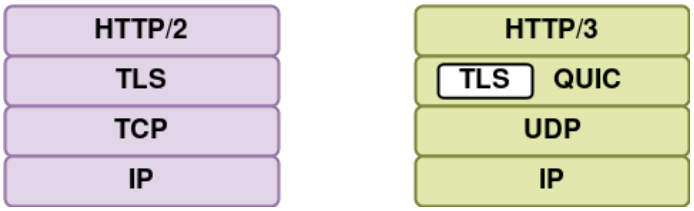

Fig. 1: Comparison between the current traditional web stacks, which use HTTP2 over TCP, and HTTP3 over QUIC. Encryption, streams, CC and recovery are built into the QUIC transport, which is delivered on top of UDP.

II. Performance over Satellite

PEPs provide acceleration for HTTP1 stacks, which require opening a new TCP connection for every request made. HTTP2 allows requests to be multiplexed over the same TCP connection (using streams), removing the main performance issues with HTTP1. However, HTTP2 is affected by head-of-line blocking: a packet loss in the underlying TCP connection affects all HTTP2 streams equally.

TLS 1.3 was released in 2019 [20]. Its main improvement over its predecessor (TLS 1.2) is a reduction in the connection setup by one RTT. For a GEO satellite, this means an improvement of more than 600 milliseconds per connection.

HTTP3 is supported by QUIC, which also provides stream support to eliminate head-of-line blocking, promising to improve performance over traditional HTTP2/TCP stacks. TLS1.3 has been built into QUIC rather than forming a separate layer.

Figure 2 presents the median time to transfer 100KB of data from a client in a satellite network using HTTP/TCP with TLS1.2 and TLS1.3, with and without PEP acceleration. This data was collected on the satellite testbed described in Section VI-A. The figure shows that the PEP accelerates TCP connections by several seconds, and, as expected, TLS1.3 further accelerates this by roughly one RTT. IETF QUIC performs better compared to TCP without a PEP, and worse that TCP with a PEP. This confirms previous findings in [7].

The split-TCP PEP works by replacing the TCP protocol with one tailored to the satellite characteristics. This tunes the way loss recovery, congestion control and flow control operate. It also effectively adapts the pattern and rate of ACKs sent over the satellite return link, ”thinning” the rate of ACKs. This thinning is not visible to an external observer, but does impact return link performance, and can impact forward link throughput. PEPs also implement two transport stacks and a flow-control/buffer management mechanism between the two, which brings down computational and buffering costs.

III. TCP Receiver Acknowledgement

TCP uses ACKs for connection establishment, reliable delivery of data, congestion control and as a clock mechanism [6]. The TCP receiver acknowledges the segments it receives using a cumulative acknowledgement number.

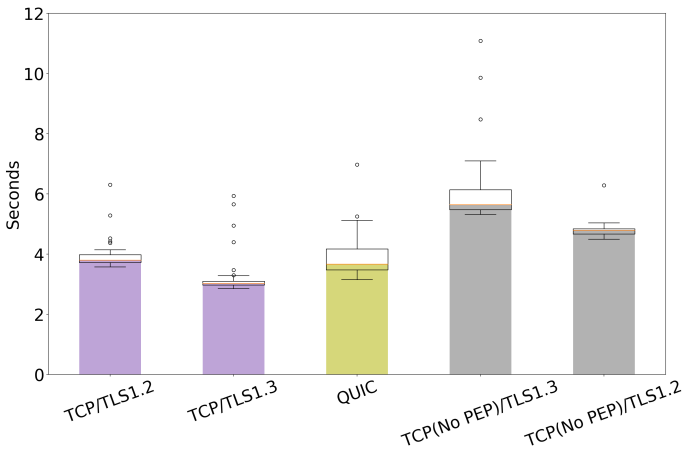

Fig. 2: Download Time for 100KB (n=50) of data using TCP and QUIC over a broadband satellite IPOS service. Although average performance is predictable, resource sharing of the return link does result in outliers, represented by circles at times of peak load.

A TCP sender is permitted to send new data when it receives an ACK for previously unacknowledged data. In this way, the sending rate is controlled by the rate of reception of ACKs, known as ACK-Clocking. A TCP receiver does not need to ACK every data segment it receives [3], instead an ACK can be held so it covers the equivalent of two times the MSS (Maximum Segment Size) of data. These are called ”delayed ACKs”. The limit of two MSS is a recommendation [3] that results in a ratio of ACK to data packets of 1:2, which is commonly implemented in TCP receivers. TCP also specifies that an ACK ”MUST be generated within 500 ms of the arrival of the first unacknowledged packet.” Many current receivers use an ack_delay value less than 500ms, e.g. 200ms or 40ms.

While a TCP connection is establishing the capacity of a path, it goes through a period of exponential growth called slow-start. During slow-start the delayed ACK behaviour can be suspended to speed up the growth rate of the cwnd, using Delayed ACKs After Slow Start (DAASS) [1].

DAASS also mitigates Stretch ACKs (ACKs for more than 2 packets), commonly encountered in the return traffic received by a TCP sender. Stretch ACKs were found to be common in the Internet [8], with ACKs covering 3 or 4 TCP segments being common in about 10% of cases. Since the introduction of Appropriate Byte Counting (ABC) [2], ACK-Clocking operates on each cumulatively acknowledged segments of data, not on individual received ACKs. This eliminates the need for implementing DAASS to grow the congestion window or mitigate stretch ACKs, although a benefit remains for paths with RTTs shorter than the TCP ACK Delay timer, where it reduces the time to open the congestion window.

TCP suspends delayed ACKs when it detects reordering, and sends an ACK for each received packet (ACK Ratio of 1:1) during loss recovery. RFC 2018 [18] added a selective acknowledgement (SACK) option to TCP. This improves forward

path recovery efficiency after loss/reordering by including SACK blocks in each ACK that describe gaps in the window of received data.

IV. Acknowledgment for a Satellite Return Path

Like TCP, QUIC is a connection-oriented transport protocol. The ACK policy used in TCP is the basis for the ACK policy for QUIC. While TCP uses ACK sequence numbers to identify bytes received and to signal the sender when data needs to be retransmitted, QUIC uses packet numbers. These monotonically increase with every packet sent, including retransmissions.

Successful transmission is indicated by receiving acknowledgements, send in ACK frames. The current QUIC specification [13] recommends sending an ACK for each alternate ACK-eliciting QUIC packet (i.e. one not purely made of ACK frames). This mimics the recommended policy used by TCP [3] and corresponds to an ACK Ratio of 1:2.

QUIC also implements an ACK Delay timeout in a manner similar to TCP [3]. The sender also communicates the maximum ACK Delay in milliseconds, with a recommended default of 25 milliseconds. This value used by both Chromium QUIC and Quicly. The receiver's ACK Delay value is shared with the peer to inform RTT calculations.

A. QUIC and Satellite

QUIC packets have a non-encrypted header [24] that allows a packet to be identified. The encrypted body contains multiple frames. Frames carry control information or data. A QUIC endpoint authenticates all headers to prevent modification. In contrast to TCP, all ACKs are encrypted, preventing use of any split-transport PEP that does not have access to the end-to-end encryption keys.

Avoiding PEPs has advantages for satellite systems. It eliminates per-flow computation costs, and the need to update a PEP as transports evolve. However, the end to end performance becomes directly reliant on the performance of QUIC itself.

Some simple ACK-thinning methods could be applied to QUIC. For example, Decimation [9], which removes small packets for a flow when a router queue builds without knowing their content. However, this does not come without potential issues, since the traffic from multiple application flows can be aggregated into a single QUIC connection and a queue of QUIC packets carries an assortment of frame types used for a variety of purposes. Hence, this method's final impact on the application using QUIC is hard to predict.

As seen in Figure 2, QUIC performs worse than TCP accelerated by a PEP. It also expects the return link to carry all unmodified QUIC ACKs. In the following subsection, we examine this ACK traffic generated by a QUIC session.

V. Understanding ACK Overhead

This section compares the efficiency of using QUIC and TCP by calculating the overhead in bytes for sending an ACK using each protocol.

A. TCP Acknowledgement Traffic

The transport layer size of a TCP ACK (without SACK blocks or other TCP options) is 20 B. The total IP and transport overhead (assuming IPv4) for a TCP delayed ACK is therefore 40 B/2*MSS (or 20 B per packet). This represents 1.33% of a 1500 B packet) [6]. The TCP Timstamp Option (TSOpt) [14] is often used to protect against TCP sequence space sequence wrap-arounds over long fat paths. This increases the TCP ACK size by 12 B (a 10 B option, plus 2 B to align to the 32 B option space), increasing the overhead to 1.7%.

After reordering is detected, most TCP receivers will generate SACK ranges (see Section III), increasing the size of ACKs. The final size depends on the pattern of loss, but will often extend to fill the option space, resulting in up to 20 B of TCP option until the loss recovery completes.

Table I presents the calculated overhead for IPv4. IPv6 would increase the size of the IP header by 20 B. Many receivers disable ACK Delay at the start of a connection using a form of (DAASS, see Section III). This results in more frequent ACKs at the start of a connection, increasing the overhead [6].

1) Analysing TCP Acknowledgement Traffic over Satellite: ACK Thinning is widely deployed in WiFi drivers as well as cable, satellite and other access technologies [6]. This intentionally removes ACKs to form Stretch-ACKs, reducing ACK rate and the corresponding volume of return path traffic. The reduction depends on the thinning policy. If it reduces ACKs to one ACK per 4 received data packets, this would reduce the total volume of ACK traffic by a factor of 2, a reduction from 1.7% to 0.9% of the total volume of traffic received. An equivalent reduction in return traffic can be provided for a satellite system using a split-TCP PEP.

B. QUIC Acknowledgement Traffic

QUIC packets sent on the forward and return paths can and will carry a variety of different types of frames to update the state of the connection making the size of return traffic vary in size. All QUIC packets have an encryption overhead, dependent on the ciphersuite used. All QUIC ciphersuites use an authentication tag that produces an output 16 B larger than their input [13].

When a QUIC receiver receives packet in order, ACK Frames will typically remain small, under 10 B, resulting in 2.4% total byte overhead per packet (assuming a QUIC packet size of 1280 B). QUIC uses a Variable-Length Integer Encoding for fields [13]. This causes the size of a QUIC ACK to grow after the first 16,384 packets have been sent, adding 2 B to encode a Packet Number to the ACK overhead. This is illustrated in Figure 3.

Following re-ordering/loss, a QUIC receiver sends ACKs with an ACK Range vector [13], similar to a TCP SACK block. Unlike TCP, the size of the ACK range can grow to fill an entire QUIC frame. While TCP limits the size of the control header to 40 B (the maximum option size), QUIC packets are not subject to a similar limit, although QUIC permits an implementation to limit this (e.g., to a few hundred bytes).

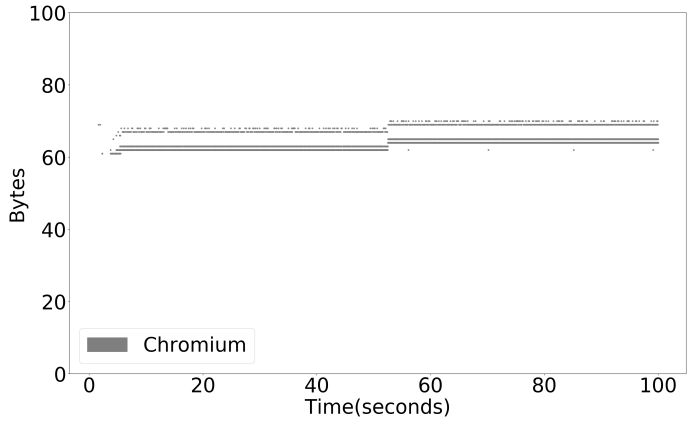

Fig. 3: Packet sizes seen on the return path during a 100 MB Chromium transfer on the satellite broadband network, after the first RTT. The step-up in size at aroung 60 seconds in the transfer corresponds to exceeding 16,000 packets sent.

	TCP cumulative ACK	TCP with loss
Min	40B/2MSS	40B+10B+2B/MSS
Max	40B+10B+2B/MSS	40B+40B+4B/MSS
Total	20B - 52B/ MSS	52B - 84B/MSS
%	1.33% - 3.46%	3.46% - 5.6%

TABLE I: Estimated overhead associated with a TCP ACK using IPv4. The percentage is calculated based on a typical header size and assuming a data packet of 1500 B, and one ACK for every 2 received packets (ACK Ratio 1:2, no DAASS).

Tables I and II summarise the estimated cost of sending ACKs. The projected overhead for QUIC is larger than for TCP. To understand the impact of QUIC traffic and ACK policy on the satellite return link subsystem, the following section examines the rate and volume of packets sent by both TCP and QUIC sessions.

VI. Experiments and Results

Experiments using a testbed were used to validate the analytical numbers in the previous section and to examine the role of ACKs as part of the total return path traffic.

	QUIC cumulative ACK	QUIC with loss
Min	28B+3B+4B+16B/2MSS	28B+3B+6B+16B/MSS
Max	28B+12B+16B+16B/MSS	Unlimited
Total	27B - 72B/MSS	53B - Unlimited/MSS
%	2.44% - 5.62%	4.14% - Unlimited

TABLE II: Estimated overhead associated with a QUIC ACK using IPv4. The percentage is calculated based on typical header size and assumes a data packet of 1280 B (ACK Ratio 1:2, no DAASS).

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%

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

A. Methodology

We analyzed the amount and composition of return traffic when transferring data with TCP and QUIC. Transfers were performed in both an emulated testbed and a satellite broadband service during February 2020, using the Quicly (commit 2f4321c draft revision 27) and Chromium (commit 53f2426 draft revision 26) implementations of QUIC.

Both QUIC implementations provide a command line client tool to run in server or client mode. `quicly` also has in-built logging for packet numbers, timestamps, RTT and congestion window information, which would otherwise be encrypted. To this, we added support for logging frame types.

Equivalent information was collected for TCP sessions by capturing packets at the client and server for the entire session using `tcpdump`. The required information was subsequently extracted from the TCP headers using `curl` and `apache2`.

1) Emulated Satellite Testbed: Experiments used a testbed to explore the detailed protocol behaviour. We emulated a dumbbell network with a client, server and a testbed router. Each host was a PC ENGINES APU2 single board computer with a 4 core 1GHz processor with 4GB of RAM and three Intel Gigabit network interfaces. All hosts run FreeBSD 12.1, the latest release at the time of the experiments.

A router emulated the network using queues configured in dummynet to emulate an 8.5 Mbit/s forward direction path and 1.5 Mbit/s return path with a total end-to-end delay of 600ms. This models the satellite broadband service described in the following section, aiming to allow for comparable results between the two. One Bandwidth Delay Product (BDP) of buffering was assigned for the forward link. The RRM was not modelled and and the return link used the default buffer size (50 packets) A fixed random packet loss rate (PLR) of 1% on the forward link was configured for some experiments.

2) Satellite Broadband Experiments: HTTPS Requests were made from a client running Debian Linux 10 connected using a GEO satellite broadband service to a server using a virtual machine at the University of Aberdeen. The RTT for the satellite path was approximately 630 ms and the nominal rate was 10/2 Mbps. We scheduled activities to avoid triggering the satellite Service Level Agreement (SLA) enforcer.

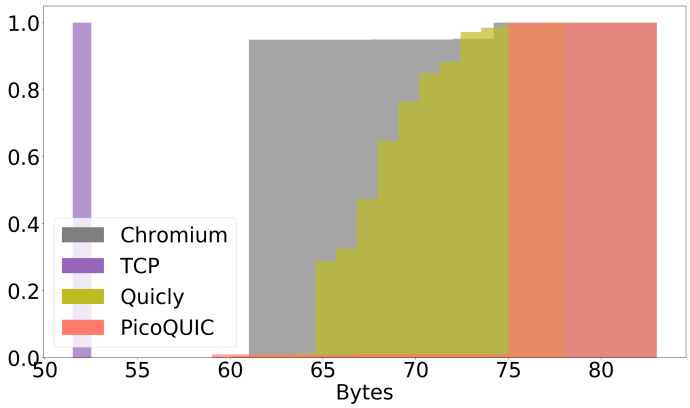

Fig. 4: CDF of packet sizes seen on the return path during a 10 MB transfer in our emulated long delay testbed, after the first RTT

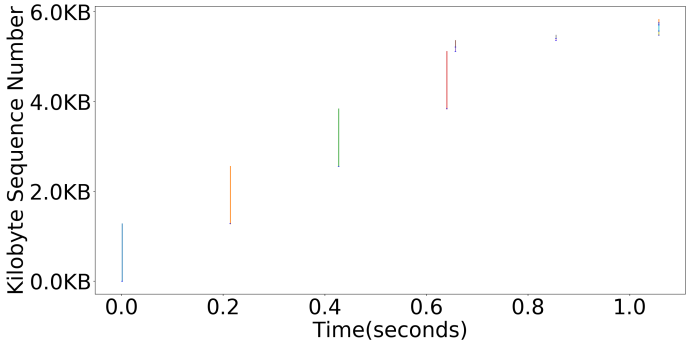

Fig. 5: Data sent by a QUIC client during the first RTT of a connection over an emulated satellite network. The plot shows 4 QUIC handshake packets, spaced 200 ms apart. Subsequent packets carry control frames and smaller in size.

B. Comparing TCP and QUIC return traffic

While TCP and QUIC have similar ACK policies, there are significant differences in the volume of data sent. Table III presents the number bytes sent on the return path for a 10 MB transfer for TCP and QUIC on the emulated satellite testbed described in Section VI-A, as well as the percentage of the total 10 MB sent on the forward path. Overall, Quicly sends more bytes than Chromium QUIC, which sends more bytes than TCP for equal transfer sizes due to a difference in the data packet size (1280 B vs 1352 B vs 1460 B).

We further investigate the size of packets sent on the return path during a 10 MB transfer. TCP, Quicly and Chromium QUIC data was obtained on the emulated testbed (Section VI-A). The PicoQUIC data was collected on a similarly configured emulated satellite testbed and provided to the authors. Figure 4 presents a CDF of the return path packet sizes for each transfer. The average TCP packet size was 52 B (40 B of IP and TCP headers and 12 B TSO), while the QUIC implementations sent packets that were 25% to 50% larger. The smallest median value for any QUIC implementation was 63 B (Chromium), while PicoQUIC's is the largest, exceeding 70 B.

The smallest possible ACK we expect QUIC to send is 51 B: 20 B (IP header) + 8 B (UDP header) + 3 B (minimum size of a short header) + 4 B (minimum size of an ACK frame) + 16 B (encryption overhead). These calculations are confirmed by observing sizes of packets on the return path, where the smallest packet size was around 60 B.

At the beginning of the transfer, QUIC exchanges a padded frame for the initial cryptographic handshake. If this fails, an initial PTO causes it to be resent after 200ms. For a satellite path, the RTT is greater than the initial PTO. The initial packet is therefore retransmitted before the handshake completes, at 0.2, 0.4, 0.6 and 0.8 seconds. We therefore omit from the CDF the first 4 frames sent at the start of a transfer, which are each 1308 B - see Figure 5 A similar effect can be seen in Figure 5 when TCP uses a 200ms initial RTO. There overhead from retransmitting the handshake is appreciable for small transfers: consuming up to 7.3% in the case of a 100 KB transfer.

We used the extensible logging system of Quicly to further investigate the composition of return traffic. Figure 6 presents a breakdown of return traffic by header and frame type, separating ACKs from other frame types. Each header is labeled with the percentage it represents from the total amount of traffic sent on the forward path, in this case, 10 MB. TCP data is obtained from a client in a satellite network. Because the PEP sends ACKs to the client, we do not see the effect of ACK Thinning, nor do we observe the TCP timestamps. These ACKs are 40 B.

We observe that actual ACK frames only account for 0.5% of forward traffic, but the total return volume is increased due to QUIC and crypto overhead, resulting in upto 3% in total. This is almost twice that for TCP, motivating a need to reconsider QUIC ACKs. Reducing the ACK volume could be achieved by using a different ACK Ratio. The effect of a 1:10 ACK Ratio is also presented in Figure 6, showing traffic volume reduced by 4 times, for a total of under 1% in a no-loss scenario. A benefit can be observed when 1% packet loss is emulated in the network.

Figure 7 presents ACK size data over time for TCP and three QUIC implementations. This data is observed at a TCP client in a satellite network, while QUIC data was collected in emulated satellite testbeds.

Using a satellite path, TCP ACKs are all 40 B (a result of a split-TCP PEP). QUIC return traffic size and pattern depends on implement choices. Chromium packets start at 60 B in size (7 B larger than the smallest ACK) and maintain their size throughout the transfer, interleaved with other larger packets (ACKs combined with other frames), seen as two parallel grey lines in the figure. Quicly's return traffic constantly varies in size (this may be due to reordering sensitivity in this implementation), while PicoQUIC's return packets are constant size (perhaps to avoid differentiation between ACKs and other types of frames).

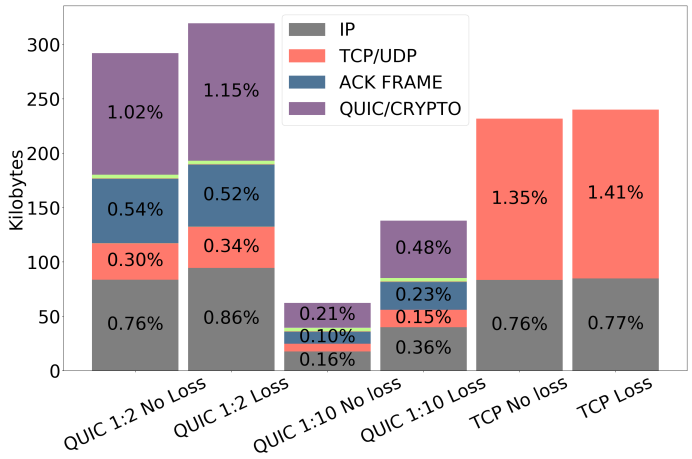

Fig. 6: Breakdown of measured return path overhead for 10 MB Quicly QUIC and TCP transfers on an emulated satellite testbed, in No Loss and 1% Loss scenarios (ACK Ratios 1:2 and 1:10 for QUIC.

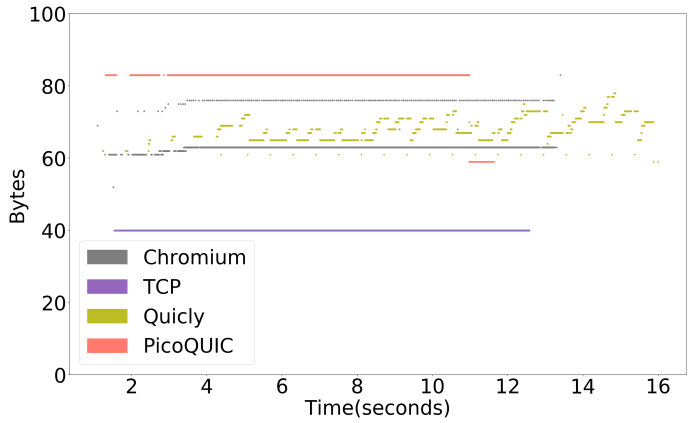

Fig. 7: Size of return packets over time seen for a 10 MB transfer, for TCP over a satellite network and three QUIC implementations in an emulated satellite testbed.

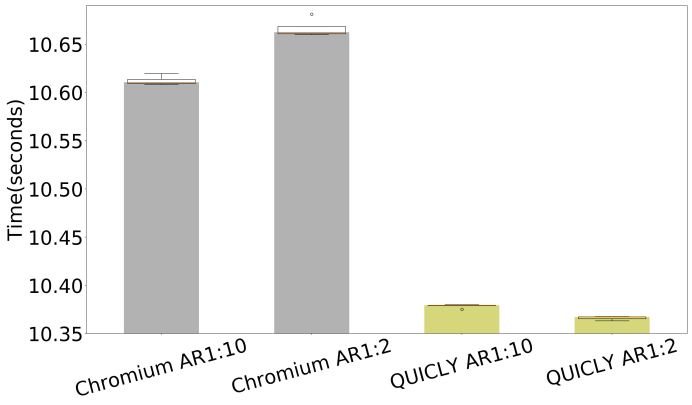

Fig. 8: Measured time to download 10 MB using Chromium and Quicly (ACK Ratio1:2 and 1:10) using an emulated short-RTT path.

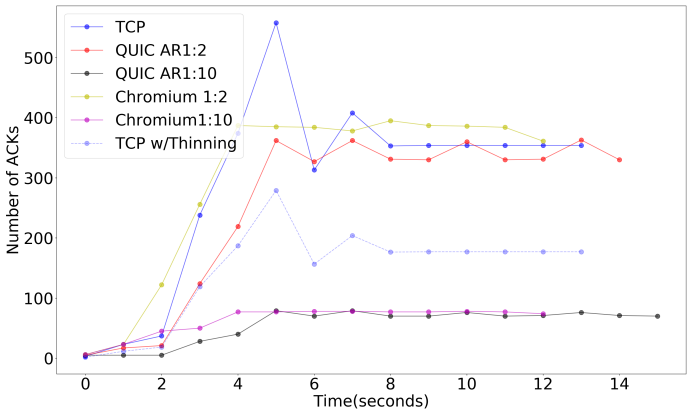

Fig. 9: Number of return packets for Quicly and Chromium using ACK Ratios of 1:2 and 1:10, measured each second. Results are shown for TCP and the estimated effect of using TCP ACK Thinning (final ACK Ratio 1:4).

C. Evaluating QUIC ACK Policy for Satellite

To study the impact of choosing a different ACK Ratio, we modified the source code of Quicly. This defined the ACK Ratio as a constant (NUM_PACKETS_BEFORE_ACK=2), which we changed to 10 for our experiments. Since the 10th of April, Quicly can also update the ACK frequency ratio using a transport frame, but changing the default still only requires modification to one line in the code.

Figure 10 presents the forward path data rate when using different ACK Ratios on a non-constrained and a very constrained satellite return path. The path asymmetry was 1:5.6 (matching the satellite service previously used). This resulted in a maximum forward packet rate of 771 pps using 1358 B QUIC packets, which filled the forward path with 8.37 Mbps of traffic (8.26 Mbps QUIC throughout). ACK congestion was observed when the capacity asymmetry was changed to 1%, causing the forward rate to be throttled to 5 Mbps. An ACK Ratio of 1:10 restored the throughput. The results shown explore Chromium, which used BBR which is a rate based congestion controller. This may need fewer ACKs/RTT [21]. The default congestion controller specified in QUIC is however not rate-based, but instead uses Reno, based on the TCP standard. Figure 9 shows the number of ACKs per second Quicly sends with comparison to Linux TCP configured to use Reno, and Chromium. For an ACK Ratio of 1:2, all transports send over 400 packets per second. The figure shows that an ACK Ratio of 1:10 results in less than 100 packets per second for both QUIC implementations, efficiently reducing not just volume, but also ACK rate. For for comparison, we again estimate performance for a simple TCP ACK Thinning mechanism.

The first packet sent by the client is a full-size padded frame, of 1308 B (Figure 5). This is larger than for initial traffic using TCP during the first RTT of a connection, but similar to performance using TCP Fast Open (TFO). For QUIC, the volume of data includes retransmissions. This increase in data

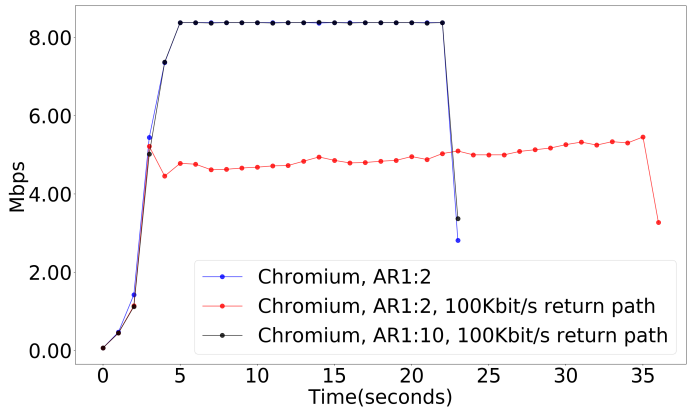

Fig. 10: Measured forward path data rate using Chromium (ACK Ratios 1:2 and 1:10), for an unconstrained return path and a path limited to 100 kbps. Rate measured each second.

volume could impact satellite return RRM, since a new QUIC flow can send much more than a typical TCP session within the first RTT. Figure 8 presents time to download 10 MB using Quicly and Chromium QUIC with an ACK Ratio of 1:2.

D. Analysis of a Modified QUIC ACK Policy

Section VI-B suggests that while the currently specified QUIC ACK policy mimics the ACK pattern of TCP, the final result is significantly poorer performance when used with GEO satellite systems. This is because the design failed to consider the implication of widely deployed PEPs that modify TCP ACK policy. This section therefore evaluates a proposal to modify QUIC's currently specified ACK policy, which is expected to benefit satellite paths.

Using an ACK Ratio of 1:10 (Figure 8) mitigates the impact of the restricted return path. The analysis in Section VI-C indicates that QUIC's operation is not negatively impacted when using an ACK Ratio of 1:10. We also evaluated the effect of this change on congestion window growth for Quicly and found no significant difference to the default ACK Ratio [6]. We therefore argue this is a safe mechanism that offers significant benefit on satellite paths.

An efficient ACK policy needs to balance the need for growing the congestion window effectively at the beginning of a transmission with the desire to ensure efficient use of the return path. In slow start, QUIC citeI-D.ietf-quic-recovery increases the congestion window by the number of bytes acknowledged when each acknowledgment is processed. This mechanism is similar to ABC. While sending more ACKs in the beginning of the transfer would mean the congestion window is increased more gradually, a DAASS-like mechanism is only beneficial for low RTT paths, where ACK Delay appreciably delays each RTT of growth. In our proposal, the ACK Delay timer ensures a minimum number of ACKs (e.g. 8) per RTT to mitigate this effect of ACK loss on RTT estimation, and aids performance for low-rate interactive applications. For a GEO path, the default value of max_ack_delay already satisfies this constraint.

The recommendation is that f slow start is a part of the congestion control method, an ACK frame should be sent for every received ack-eliciting packet for the first 100 received packets if max_ack_delay (default 25 ms) has passed since the oldest unacknowledged data was received. This ensures Stretch ACKs do not impact the initial rate of congestion window growth, and could particularly benefit paths with an RTT less than the default ACK Delay.

VII. Discussion

Our results show that the performance of QUIC is similar to that of TCP using HTTP2 across a satellite path when using a well-tuned PEP, however, there are critical areas in which performance can be better. The focus of this paper has been on the ACK traffic generated by QUIC when used for downloads over a forward satellite path.

Section VI-B observed an increase in the size of initial packets compared to TCP, and a significant increase in the volume and number of packets sent in the first 2 RTTs of a connection. This will have implications on RRM design, because this control data is sent before a BoD system has had time to adjust to starting a new connection. Simple solutions, such as pre-allocating a small return link timeslot to each active terminal, are less efficient for a larger volume of traffic.

As currently specified, QUIC generates slightly more ACK packets than TCP, but consumes significantly more return link capacity. This can increase resource usage of the return channel, and may constrain forward throughput. This can impact the performance of other sessions sharing an RRM-controlled return link.

We argue that for encrypted transports such as QUIC, a suitable ACK Ratio has to be set by the endpoints implementing the protocol. A change to the default is vital because the sender and receiver have no information about the "cost" of sending an ACK over a path including a satellite system. We showed that satellite systems operating at 10 Mbps benefit from an ACK Ratio of 1:10. Even in next generation 5G systems [17] there is still a significant cost for uplink transmission compared to downlink, and a high ACK rate impacts either the user itself, or others users that share the same capacity pool. Implementing this change would also disincentivise mitigations such as ACK Decimation [9].

Higher rates of transmission or greater asymmetry may benefit from even less frequent ACKs (e.g. an ACK Ratio of 1:100). Such scenarios may be common to other technologies, not just satellite systems, and could present opportunities for negotiating a more asymmetric ACK Ratio. For example, to reduce the processing cost at endpoints [12] (important as packet rates increase: an aggregate data rate of 1 Gbps results in 5000 ACK/sec using an ACK Ratio of 1:2.) This can also be used to adjust to ACK Policy following congestion on the return path [12]. Although QUIC has mechanisms that could be used by the sender to detect ACK congestion, there is no currently no specified mechanism to tell the receiver to adapt the ACK rate [1]). Although this suggests it may be attractive to

[1]https://github.com/quicwg/base-drafts/issues/1978

use a ratio more asymmetric than 1:10, this raises additional questions. Whilst a choice of 1:10 is balanced by the existing need for QUIC to pace traffic (and an IW of 10 requires such pacing to be appropriate for bursts of the order of 10 packets), using a more asymmetric value could negatively interact with congestion control and loss recovery.

One method to permit a sender to influence the ACK policy used by a receiver is to use a protocol mechanism that dynamically updates the QUIC transport parameters [13]. This could control the number of ACK-eliciting packets accumulated into an ACK [12] over the lifetime of a connection. The negotiation of the mechanisms and the tuning of parameters for satellite and non-satellite scenarios are an area for future experimentation.

VIII. QUIC Flow Control

Protocols also need to provide mechanisms to prevent a receiver being overwhelmed with data when a sender and path support a higher rate than can be sustained at the receiver. In TCP, this is known as the receiver window mechanism. In QUIC this is known as flow control. QUIC uses a credit-based mechanism rather than the window based mechanism in TCP.

QUIC as a multi-streaming protocol has two levels that flow control credits are issued, in total for the connection and per stream flow credits. Connection flow control is intended to stop a sender from exhausting a receivers buffer, while stream flow control is intended to limit the amount of the connection a single stream can use.

The larger RTT for a satellite path increases the time taken to respond to any control message, and hence can place additional demand on these methods. When using a satellite system, the large RTT results in a need for buffering that can significantly impact the top speed. This is one motivation for a more detailed analysis of application performance using QUIC with actual web workloads [19].

IX. Conclusion

This paper has analysed the performance of the latest specification of the QUIC protocol, with an emphasis on how QUIC return traffic interacts with a satellite system. It has evaluated the design choices of QUIC ACK generation and proposed an alternate ACK policy to benefit asymmetric paths. We argue that using a more suitable ACK Ratio value would benefit the satellite use-case and other paths with appreciable asymmetry (cable networks, mobile cellular, WiFi). We plan continue to evaluate other aspects of the QUIC protocol with further experimentation across a range of satellite scenarios.

The IETF QUIC specifications are set to be published in 2020. A small number of players (Google, Cloudflare etc) who control a very large share of web traffic have already deployed QUIC. With the expected growth in the volume of encrypted QUIC traffic, satellite systems will have to adapt to support a mixture of both TCP applications (which will continue to need PEP infrastructure), as well as an increasing amount of QUIC web traffic. The transition to QUIC will likely be accompanied by an increased range of applications supported using QUIC. The change to a fully-encrypted transport will also dictate changes to operational practice for satellite systems.

Acknowledgements

The MTAILS CCN project has received funding from the European Space Agency under Contract No. 4000122992. the authors thank Nicholas Kuhn for discussion and data relating to the performance of PicoQUIC over an emulated satellite link.

References

[1] M. Allman. On the generation and use of TCP acknowledgments. *ACM SIGCOMM CCR*, 28(5), 1998.
[2] M. Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465 (Experimental), Feb. 2003.
[3] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681, Sept. 2009.
[4] C. Caini, R. Firrincieli, and D. Lacamera. PEPsal: a Performance Enhancing Proxy designed for TCP satellite connections. In *IEEE 63rd Vehicular Technology Conference*, 2006.
[5] L. Caviglione et al. A deep analysis on future web technologies and protocols over broadband GEO satellite networks. *Int. J. Satellite Communications and Networking*, 33(5), 2015.
[6] A. Custura, T. Jones, and G. Fairhurst. Rethinking ACKs at the transport layer. In *FIT*, June 2020.
[7] J. Deutschmann, K. Hielscher, and R. German. Satellite Internet Performance Measurements. In *NetSys*, 2019.
[8] H. Ding and M. Rabinovich. TCP stretch acknowledgements and timestamps: findings and implications for passive RTT measurement. *ACM SIGCOMM CCR*, 45(3), 2015.
[9] H. B. et al. TCP Performance Implications of Network Path Asymmetry. RFC 3449, Dec. 2002.
[10] G. Fairhurst and A. Yun. Design of the DVB-RCS2 higher layer satellite architecture. *Int. J. Satellite Communications and Networking*, 31(5), 2013.
[11] J. Iyengar and I. Swett. QUIC Loss Detection and Congestion Control, July 2019. IETF Work in Progress.
[12] J. Iyengar and I. Swett. Sender Control of Acknowledgement Delays in QUIC, 2020. IETF Work in Progress.
[13] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport, Jan. 2017. IETF Work in Progress.
[14] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323, May 1992.
[15] A. Langley et al. The quic transport protocol: Design and internet-scale deployment. In *ACM SIGGCOMM*, 2017.
[16] F. Lawas-Grodek et al. SCPS-TP, TCP and rate-based protocol evaluation for high delay, error prone links. In *SpaceOps Conference*, 2002.
[17] K. Liolis et al. Use cases and scenarios of 5G integrated satellite-terrestrial networks for enhanced mobile broadband: The SaT5G approach. *Int. J. Satellite Communications and Networking*, 37(2), 2019.
[18] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018, Oct. 1996.
[19] A. Mohideen et al. Evaluating the impact of transport mechanisms on web performance for effective web access. *Journal of Network and Computer Applications*, 137, 2019.
[20] E. Rescorla. The transport layer security (tls) protocol version 1.3. Internet Requests for Comments, Aug. 2018.
[21] D. Scholz et al. Towards a Deeper Understanding of TCP BBR Congestion Control. In *IFIP*, 2018.
[22] R. Secchi, A. C. Mohideen, and G. Fairhurst. Performance analysis of next generation web access via satellite. *Int. J. Satellite Communications and Networking*, 36(1), 2018.
[23] M. Sooriyabandara and G. Fairhurst. Dynamics of TCP over BoD satellite networks. *Int. J. Satellite Communications and Networking*, 21(4-5), 2003.
[24] M. Thomson. Version-Independent Properties of QUIC, June 2020. IETF Work in Progress.
[25] M. Thomson and S. Turner. Using TLS to Secure QUIC, Jul. 2019. IETF Work in Progress.