Received: 27 June 2022 | Revised: 29 August 2022 | Accepted: 6 October 2022

DOI: 10.1002/sat.1466

ORIGINAL PAPER

WILEY

Reducing the acknowledgement frequency in IETF QUIC

Ana Custura | Tom Jones | Raffaello Secchi | Gorry Fairhurst

School of Engineering, University of Aberdeen, Aberdeen, UK

Correspondence
Ana Custura, School of Engineering, University of Aberdeen, Aberdeen, UK.
Email: ana@erg.abdn.ac.uk

Funding information
Ana Custura is funded by the School of Engineering at the University of Aberdeen to perform transport protocol research on paths with different characteristics. Tom Jones is funded by the European Space Agency to research the evolution and impact of QUIC on satellite systems.

Summary

Satellite networks usually use in-network methods (such as Performance Enhancing Proxies for TCP) to adapt the transport to the characteristics of the forward and return paths. QUIC is a transport protocol that prevents the use of in–network methods. This paper explores the use of the recently–standardised IETF QUIC protocol with a focus on the implications on performance when using different acknowledgement policies to reduce the number of packets and volume of bytes sent on the return path. Our analysis evaluates a set of ACK policies for three IETF QUIC implementations, examining performance over cellular, terrestrial and satellite networks. It shows that QUIC performance can be maintained even when sending fewer acknowledgements, and recommends a new QUIC acknowledgement policy that adapts QUIC's ACK Delay value based on the path RTT to ensure timely feedback. The resulting policy is shown to reduce the volume/rate of traffic sent on the return path and associated processing costs in endpoints, without sacrificing throughput.

KEYWORDS
ACK, QUIC, satellite

1 | INTRODUCTION

The recently-standardised QUIC transport protocol[1] is set to replace the Transmission Control Protocol (TCP) stack for a wide variety of applications. QUIC has been developed to address the deficiencies of TCP, which can no longer be extended without breaking compatibility with legacy systems.[2] QUIC is a connection-oriented UDP datagram protocol, which brings faster connection opening, the ability to combine multiple streams in a single congestion-controlled flow, better integration with TLS (Transport Layer Security), and mitigation of head-of-line blocking in concurrent streams. Peers in a QUIC connection communicate by sending packets composed of frames, which carry data and control information. Apart from a small fixed header, the entirety of a QUIC packet is encrypted and authenticated.

Like other transport protocols, QUIC uses Acknowledgements (ACKs) and offers improvements in the way ACKs are used. This paper focuses on the policy used by a QUIC receiver, which determines when a receiver generates an ACK frame for the data it has received.

Many important consumer technologies, such as cellular and satellite broadband, either have a lower layer that is shared (e.g., WiFi) or exhibit some form of path asymmetry, where the capacity or transmit cost in one direction differs significantly from the other. ACK forwarding incurs costs as it impacts power consumption and link usage for devices that transmit packets over such networks.[3] In broadband satellite and cellular systems, transmission of ACKs incurs cost to the user/operator of the radio segment, usually controlled by the Radio Resource Management (RRM),[4] and consumption of link resources can impact other users that share the link or are allocated capacity from the same resource pool. To mitigate this for TCP satellite and other networks with asymmetric links, various methods or in-network devices (such as split-TCP Performance Enhancing Proxies [PEPs]) are used. However, these methods cannot be used by QUIC due to its always-on

privacy features. Decreasing the ACK rate can also reduce the CPU utilisation overhead associated with QUIC packet transmission and reception for endpoints in any network.[5]

The ACK policy specified by QUIC[1] recommends sending one ACK for every two received QUIC packets, which we refer to as an ACK: data Ratio of 2, or AR 2. RFC 9000 also specifies an *ACK Delay* of 25 ms to ensure ACKs are sent when data rates are low. There is interest in alternative ACK policies for QUIC: several QUIC implementations are already adopting an AR greater than 2, and in July 2020, the IETF QUIC WG started defining an extension that allows a sender to transmit an ACK Frequency (AF) Frame[6] to request the receiver to change its ACK policy. There is, nonetheless, no previous analysis of the expected performance, or of the impact on other QUIC mechanisms, like pacing and reordering detection, that become important when sending ACKs less frequently. This paper addresses the gap of understanding these interactions on networks with a range of characteristics, including those with radio segments or asymmetry. This exploration of different ACK policies takes into account the relationship between the path *Round-Trip Time* (RTT) and the *ACK Delay* value, showing that the interaction of these two values influences the total number of ACKs. It also considers a data-based policy (based on AR), and a time-based policy (based on *ACK Delay*)—generating an ACK after a specified time has passed.

This paper builds on previous work[7] by the authors, where AR 10 was evaluated in GEO satellite networks. An AR of 10 was chosen as a starting point given the mounting evidence that Internet transport protocols can sustain an initial burst of 10 packets.[8] Here we extend this work by examining the effect of an AR of 20 and 100, on a variety of network paths. We also explore how the AR can influence throughput at the start of a QUIC connection and in steady-state, with and without packet loss. Finally, we discuss the implications of increasing the AR for QUIC and identify a set of prerequisites for safely doing so without impacting flow continuity.

The remainder of the paper is organised as follows. Section 2 discusses the implications of encrypted QUIC ACKs in wireless/asymmetric network segments, which motivates this paper. Section 3 provides an overview of ACKs and their application in TCP and QUIC. Section 4 outlines the methodology for this study, while Section 5 presents a survey of the use of a default AR and the prevalence of the AF Frame, and includes experimental results with Firefox, Chromium and Fastly QUIC in DSL, cellular and satellite networks. The results and their implications for radio network operators are discussed in Section 6 and our findings and recommendations are summarised in Section 7.

2 | MOTIVATION

Many consumer broadband technologies that include radio network segments are also asymmetric, where the path characteristics differ in the forward and the return path.[9] Operators of networks with asymmetry have often resorted to utilising adaptation methods or deploying network devices that adapt the transport protocol to the characteristics of specific link technologies. These methods often reduce the ACK rate and ACK volume. This is important because ACK transmission can be costly to the user and operator. This has been used in WiFi[7] and the return link between a satellite user terminal and the gateway.

Transport adaptation methods have been traditionally deployed for TCP. TCP is a bidirectional transport using ACKs to receive feedback about the connection progress, confirm which packets have been received, and estimate the path RTT.[10] Above TCP, applications can use the HTTP protocol to access the world wide web (Figure 1). In combination with HTTP/2, TLS provides security by encrypting the HTTP payload. However, the transport headers remain unencrypted, allowing network devices to adapt TCP to different link characteristics. HTTP/3[11] (published in June 2022) is set to succeed HTTP/2 and replaces TCP with QUIC,[1] incorporating TLS 1.3 (specified in RFC 8446).

An important characteristic of QUIC is the use of encryption, both for the transport and the application-layer header. This prevents network devices from accessing the transport headers and adapting the transport protocol.[12] For example, ACK Thinning in wireless links filters redundant TCP ACKs at a link interface, taking advantage of cumulative acknowledgements to reduce the ACK rate.[13,14] Although the authors have previously shown that QUIC's default ACK policy can result in throughput limitation in asymmetric network scenarios,[7] this method for ACK rate reduction cannot be used by QUIC.

Asymmetric networks can use header compression to reduce the size of packets over a network link, such as RObust Header Compression (ROHC)[15] or IP Header Compression (IPHC).[16] This functions by observing and then removing redundant header fields from packets belonging to a

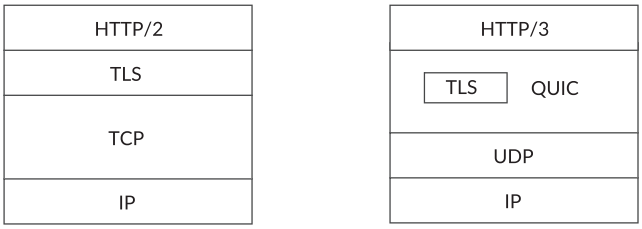


FIGURE 1 HTTP/2 over TCP and HTTP/3 over QUIC

flow at ingress and restoring these at the egress of the network segment. Methods are designed to be robust to loss.[17,12] While both the TCP and IP headers are compressible, a compressor observing a QUIC flow can only reduce the size of the IP and UDP headers. QUIC needs a UDP compression profile, and will typically introduce an extra header field for synchronisation[15] because the transport headers cannot be decrypted. The encrypted parts of the QUIC packet are not compressible, so compression can only reduce the volume of network headers but is unable to reduce the ACK rate.

Transport protocol proxies (PEPs) have been widely-deployed in GEO satellite communications systems and some cellular networks. Although a PEP can operate at various protocol layers to enhance the TCP performance,[18] with the emergence of HTTP/2 with TLS, the main contribution of PEPs is in the form of TCP transport acceleration. This remains effective, because HTTP/2 already brings benefits for a satellite user.[19,20] A split-TCP PEP (e.g., Caini et al.[21]) operates by observing TCP headers to separate the end-to-end TCP session into a series of concatenated connections, enhancing loss recovery, congestion control and flow control. The protocol used on an enhanced link can suppress ACK-only packets, which contributes to reducing capacity requirements with minimal impact on the continuity of the flow.[21] The use of TLS by QUIC prevents any enhancement in the network (including ACK Thinning and PEP). The UDP packets carrying QUIC traffic are typically forwarded by a PEP without modification, which further motivates this work to reduce the ACK rate/volume of QUIC. There are also more general benefits from reducing the ACK rate. For TCP, ACK processing can consume as much as 20% of CPU cycles in server applications.[5,22] Many high-rate network cards use Large Receive Offload (LRO) to reduce per-packet receive processing, with hardware-agnostic methods (such as Generic Receiver Offload, GRO) also providing a similar function.[23] These benefits could also be realised if QUIC reduces its ACK rate.

3 | BACKGROUND

Proposals to change the ACK policy of transport protocols to minimise ACK rate and volume require care; sending too few ACKs can negatively interact with other transport mechanisms, such as Congestion Control (CC) and loss recovery, and result in traffic stalls and reduced performance. This section, therefore, provides a background to the role of ACKs in both TCP and QUIC.

3.1 | TCP ACKs and congestion control

TCP uses ACKs for connection establishment, reliable data delivery and CC.[10] A receiver does not need to acknowledge every received segment.[24] Instead, a "delayed ACK" can cover up to two times the Maximum Segment Size of data. This recommendation in RFC 5681 results in the commonly implemented policy of AR 2. TCP specifies an *ACK Delay* of 500 milliseconds to accommodate low data rates, although many current receivers use a value less than 500 ms, for example, 200 or 40 ms.[25]

While a TCP connection is determining the path capacity, a Reno CC uses ACK-Clocking (whereby a sender sends new data each time it receives an ACK covering previously unacknowledged data) to exponentially grow the number of packets sent per RTT, known as the congestion window (`cwnd`). This multiplicative increase phase is called *slow-start*. RFC 3465 modified TCP accounting to be byte-based rather than packet-based. Before this, ACK-Clocking operated on each received ACK and so delayed ACKs could be suspended during slow-start to speed up `cwnd` growth,[26] using a mechanism called DAASS (Delayed ACKs After Slow-Start). In current TCP implementations, ACK-Clocking operates on cumulatively-acknowledged bytes, partly eliminating the need for this mechanism. On network paths with appreciable delay (e.g., GEO satellite paths), the `cwnd` can grow proportional to the Bandwidth-Delay Product, increasing the number of ACKs per RTT.

CC methods such as Cubic adapt how the `cwnd` grows. Bottleneck bandwidth and round-trip propagation time (BBR)[27] is an alternate rate-based CC method that does not rely on ACK-Clocking. Although deployed, this method is still evolving and is not yet standardised.

TCP senders use information in ACKs to determine which packets have been lost. At the sender, received ACKs that indicate reordering can be used to trigger a fast retransmission. To avoid unnecessary delay, a TCP receiver does not defer ACK transmission after it detects reordered packets. Instead, it sends an ACK for each received packet (AR 1) as soon as it receives the first packet out of order until the end of loss recovery. RFC 2018 extended TCP by adding a selective acknowledgement (SACK) option. One or more SACK blocks are included in an ACK to describe gaps in the received window of data. This can allow a TCP sender to improve the efficiency after loss/reordering.

Methods described in Section 2, such as ACK Thinning, could result in one ACK covering more than two packets. This could cause the release of multiple packets into the network at line rate, potentially exceeding path capacity or causing congestion. TCP modifications can mitigate this by using sender pacing and timer-based retransmissions,[28] and by adapting the CC algorithms.[29]

3.2 | QUIC ACK frames and congestion control

QUIC transmits information in QUIC packets, that are encapsulated in UDP datagrams for transmission across a network. The payload of each QUIC packet is composed of QUIC frames, which encode control information and data. QUIC is extensible with new frame types, enabling

enhancements and extensions to the protocol as new uses are explored. While TCP uses sequence numbers to track bytes successfully received and to infer when packets are lost, QUIC uses a monotonically increasing Packet Number (PN). A PN is used only once each time a packet is sent, and this simple sequence number space allows QUIC to resolve the transmission ambiguity present in TCP.[1] QUIC can use the CC mechanisms specified for TCP. It also includes protocol features that allow CC mechanisms not possible in TCP due to lack of header space for expansion.

QUIC receivers report successfully received packets using ACK frames that track the largest received PN. To improve the response to loss events, ACK frames contain ACK ranges that encode successfully received packets and indicate which packets have been lost (similar to a TCP SACK). When a loss is detected, the sender can choose to retransmit frames in lost packets using a new packet identified by a new PN. To assist in loss detection and recovery QUIC recommends sending an ACK without delay whenever a PN is received out-of-order (as in Allman et al.[24]). Unless a loss is detected, an ACK arriving at a QUIC sender does not trigger the transmission of a new packet (as in ACK-Clocked TCP), instead packets are paced out by the sender at the estimated connection rate, making QUIC ACK reception and packet transmission two independent processes.

The QUIC specification[1] recommends sending an ACK frame every other packet (i.e., AR 2), mirroring the ACK policy of TCP.[24] QUIC also specifies a maximum *ACK Delay* with a recommended default of 25 ms. QUIC ACK-only packets (containing only ACK frames) are larger than their TCP counterparts (as shown in Section 5). In addition to the ACK frames, QUIC also sends other control frames throughout the lifetime of a connection. The additional frames vary in size, including full-size padded cryptographic handshake packets at the beginning of a connection, flow control updates (updated per-stream and per-connection), connection migration updates and keep-alives.[1]

3.3 | Previous work

A 2020 code study[30] on QUIC* implementation variability found that only two of the twelve studied QUIC implementations consistently used the recommended ACK Ratio (AR 2), with others using an AR between 1 and 10. This was attributed to receiver implementations reading up to ten or more packets at a time from the network socket (impacted by the pacing strategy[30]). An AR of 10 has been reported to increase computational efficiency in all networks when QUIC ACKs are processed in user-space.[5] Yang et al[31] further found that ACKs and packet reordering have a significant impact on CPU utilisation, and explored the applicability of moving these functions into hardware.

A study of the benefit of reducing the ACK rate for TCP[32] described benefits for wireless and asymmetric networks and sought to determine whether a lower ACK rate could be used in all networks without sacrificing performance. It noted that TCP's delayed ACK policy introduced a transmission delay compared to acknowledging every segment and proposed sending an ACK every RTT/4 (equivalent to an *ACK Delay* equal to RTT/4). It noted the need to modify a TCP sender to ensure no segments are clocked-out upon receiving a packet with a SACK option (as long as the sender is not in loss recovery) and to further delay the Fast Retransmit to prevent its loss. It also noted that pacing introduces further delay and requires a mechanism to reduce the sending rate that is not ACK-based. Even if in their study the interactions between RTT measurement and *ACK Delay*[32] were not considered, we show in Section 6 that sending an ACK 4 times/RTT forms an excellent starting point for an ACK policy for QUIC. This becomes more troublesome as the RTT or *cwnd* grows. The greater the value of RTT/4, the greater the delay in reducing the sending rate after detecting congestion. For *cwnd*s of over 100 packets, sending an ACK every RTT/4 causes bursts and introduces a reliance on pacing. Therefore, we will consider RTT/4 a lower bound.

In a previous work,[10] we modified the QUIC receiver to generate an ACK every *N* packets (an AR of *N*). This modification could be implemented without changing the sender. This study allowed us to examine the performance of two IETF QUIC implementations using an AR of 10 over a satellite path.[7] We showed that an AR of 10 improves performance without negatively affecting the CC when the RTT is larger than the configured *ACK Delay* (25 ms by default). This work was extended by two other studies[33,34] that used a QUIC implementation modified with an AR of 10 in network simulations to analyse a range of RTTs. The studies found that an AR of 10 improves performance in presence of a satellite RTT or asymmetry, as we also observed in previous experiments using Quicly and Chromium QUIC on a simulated satellite testbed.[7,10] The present work confirms these findings with application workloads over real networks.

Volodina et al[34] also found that an AR of 10 could decrease performance for RTTs of under 20 ms, over a path with 1% packet loss, especially when using Reno CC. It is unclear if their implementation included a policy to immediately send an ACK when reordering was detected (as now specified in Iyengar and Swett[35]), which mitigates this effect. Throughput limitation is an edge effect present with a small RTT and *cwnd*, which motivates a need to periodically acknowledge the window when a sender is *cwnd*-limited (e.g., at the start of connection). Our previous results suggest this can be mitigated by ensuring an ACK is sent at least every RTT/4.[7,10] This approach is also confirmed by Volodina et al,[34] showing that it minimises the reported decrease in throughput. In the present paper, we explore how this can be realised in a sender-based method where the RTT is known, rather than in the receiver-based method described in Custura et al.[7]

*Our paper exclusively focuses on the recently-standardised QUIC IETF protocol.[1]

The IETF work-in-progress to specify a format for a QUIC AF Frame allows both the AR and *ACK Delay* to be set. This will allow new methods to adapt these values for each connection. A recent survey in April 2022 identified four implementations that had already started using this AF Frame.

The present work builds on our previous contributions with:

- A new study evaluating how ACK policies perform with different loss and RTT regimes, using the Quicly implementation across DSL, 4G and Satellite non-simulated networks;
- New insights into client traffic composition for the now mature implementations of Firefox QUIC and Chromium QUIC with an application workload, video over HTTP/3, in non-simulated networks;
- Better understanding of the use of the AF Frame[6] in QUIC implementations;
- Discussion of header compression implications, not previously considered;
- Understanding of the interaction between the *ACK Delay* value and the RTT, and its impact on CC and on the total number of ACKs.

4 | METHODOLOGY

4.1 | Video streaming and AR experiments

To explore the effect of the AR with real QUIC workloads, video stream requests over HTTP/3 were made from Firefox version 78 and Chromium version 90 from a Debian 11 machine, over UK-operated satellite, 4G and VDSL links.

We captured a set of 80 packet traces and used Pyshark to decrypt and post-process the captures. Several other captures could not be decrypted and were not included due to a lack of support in Pyshark for recovering initial keys from multiple non-consecutive crypto frames. The requests were made to YouTube and Facebook's video CDN, which are known to deploy services over QUIC. Measured capacity, delay and packet loss for the networks are shown in Table 1. These networks were chosen in part because they have different RTTs with respect to the default *ACK Delay* value of 25 ms. Video streaming was selected to study how ACK policies change over the lifetime of a connection. Table 1 also shows the relationship between *ACK Delay* and the RTT of each network studied. The initial handshake and subsequent QUIC traffic are captured in *pcap* format, and the decryption keys are exported from each browser session allowing the QUIC traffic in the captures to be decrypted and analysed. The browser cache is cleared between each capture and an ad-blocker was used to disable ads interrupting the video content. These packet captures will be made available as open data for use by the research community.[†]

The same experimental setup was used to understand the effect of using different ACK policies over paths with loss, with some modifications. We used a different QUIC implementation (Quicly) that allowed us to control both client and server at the same time, and to modify the ACK policy (AR and *ACK Delay*). Quicly does not implement an HTTP/3 server, therefore we chose a 1 MB transfer as the workload. This size of the transfer is sufficiently small that the experiment results remain unchanged when using higher capacity links. This is because the performance for any link with a capacity higher than 5.3 Mbps and 600 ms delay (where IW = 10 and MSS = 1460 B) is dominated by the effects of slow-start. This test was repeated 50 times from a Quicly client connecting from each network in Table 1, using multiple ARs. The transfers were captured and then decrypted. To study the effects of loss, the test was repeated with 1% symmetric packet loss on the server side introduced by the *netem* emulator, as shown in Figure 2.

To compare the effects of different ARs on the number of bytes generated, we also performed several 1 MB web transfers from a Firefox and Chrome client to a Cloudflare front-end server. Using Firefox and Chrome generated slightly more bytes from the client for the same AR than Quicly. This difference allows us to evaluate the contribution of HTTP traffic to the total amount of traffic generated by the client.

TABLE 1 Median measured characteristics of the network paths utilised for video and ACK Ratio QUIC experiments

	Service	Fw link	Ret link	RTT	Packet loss	QUIC default ACK Delay as function of RTT
Satellite	IPOS terminal using Hylas 1	10 Mbit/s	2 Mbit/s	638 ms	0.001%	ACK Delay= RTT/25
4G MNO	EE 4G UK	20 Mbit/s	10 Mbit/s	95 ms	0.1%	ACK Delay= RTT/4
VDSL	Plusnet fibre broadband	40 Mbit/s	10 Mbit/s	30 ms	0.5%	ACK Delay= RTT

[†]To be published in https://github.com/uoaerg.

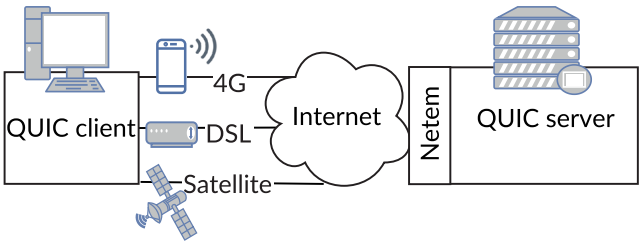


FIGURE 2 Testbed used for experiments in loss scenarios. Clients connect to the Internet through either satellite, 4G, or DSL to a server run by the researchers at the University of Aberdeen. The *netem* emulator is used to introduce server-side packet loss

TABLE 2 An implementation survey showing the default CC, AR and whether the implementation supports the AF Frame. Where the median ACK size is known to the authors from running the implementation or analysing captures in no-loss scenarios, this is recorded in the last column

Implementation	Default CC	Parses AF	Uses AF	Default AR	ACK size[2]
quicly	Reno	Y	Y	2	67
neqo	Reno	Y	Y	2	63
chrome	Cubic[1]	Y		2, then 10	66
picoquic	Reno[1]	Y	Y	≥ 2, based on *cwnd*	75
quic-go	Reno	N	N	2	
ngtcp2	Cubic	N	N	2	
aioquic	Reno	N	N	2-8	
quiche	Cubic	N	N	1-37	
mvfst	Cubic[1]	Y	N	10	
lsquic	Cubic[1]	Y		2 or 1/RTT	
nginx	N/A	N	N	2	
quant	Reno	N	N	2	
kwik	Reno	N	N	2	
msquic	Cubic	Y	Y	2	

[1]Implementations that also support BBR CC.
[2]Median size of ACKs measured by the authors using the default implementation parameters.

4.2 | Survey of QUIC implementations for ACK policies

To determine the ACK policy used by QUIC implementations, we performed a source code survey of the Open Source implementations that are actively evaluated by the QUIC interop runner.[36] We determine that an implementation supports the AF Frame by verifying that the frame is not just decoded, but that the state machine variables are modified by receipt of the frame. Specifically, we look to see if the frame parsing code would lead to changes in the ACK timer mechanisms. To determine if an implementation sends the AF Frame we search for any code that would attempt to encode the AF Frame or transport parameter when sending. Table 2 shows our findings from analysing the source code for these implementations.

5 | RESULTS

5.1 | ACK Ratio and ACK frequency frame support

Several distinct ACK policies can be differentiated among current QUIC implementations. Where the median ACK size is known to the authors from running the implementation or analysing captures in no-loss scenarios, this is recorded in the last column of Table 2. A majority of implementations seek to use an AR of at least 2, while some others use RTT-based or *cwnd*-based methods to set the AR. Implementations often read multiple packets at a time from a socket, occasionally resulting in ACKs[30] covering multiple packets. Some intentionally use a larger AR. Google Chrome currently uses an AR of 2 for the first 100 packets and an AR of 10 for subsequent packets. This is based on the assumption

that after 100 packets, the *cwnd* has already grown enough to mitigate the negative effects addressed by DAASS. On the server side, Google Chrome and Facebook Mvfst use a lower bound of 10 for the AR. Since the AR can influence the CC operation, we also examined the default CC used by current QUIC implementations and found that several use Cubic rather than Reno (Quiche, Mvfst, Msquic and Chrome). CC relies on ACK signalling to different degrees: Reno requires ACKs to maintain its sending rate, whereas for Cubic and BBR[27] the sending rate is not tightly coupled to the reception of ACKs. We also examine support for the AF Frame. Several surveyed implementations have implemented the AF Frame, including Firefox's Neqo. However, support is not yet widespread and has not increased significantly since 2020,[30] although this is expected to change as soon as the QUIC specification will be widely deployed.

5.2 | Client traffic composition

The minimum QUIC ACK size is 51 Bytes (B)[7] according to Iyengar and Thomson.[1] It comprises an 8 B UDP header, a 20 B IP header, a 3 B QUIC short header, a 4 B ACK frame and a 16 B authentication tag. However, ACK packet sizes can vary between implementations due to variations in padding, QUIC header length and variable length encoding. Padding of QUIC packets, including ACK packets, has also been proposed as a mechanism to prevent traffic analysis,[1] although we did not observe this in our tests. The last column of table 2 reports the median ACK size for each client in our experiments. We note that QUIC clients used sizes of 60-80 B, which is up to twice the minimum of 40 B for a TCP ACK—although the latter is unencrypted. Moreover, the variable length encoding of fields causes the size of a QUIC ACK to increase as a connection sends more data. This means QUIC will generate up to twice the overhead of TCP for a transfer when using the same AR.

We estimate that the minimum ACK overhead for TCP with 1% loss is 52 B for each packet received. This grows up to 84 B when a packet contains a maximum of three SACK ranges. We also estimated an equivalent minimum size for QUIC as 51 B[7] noting that a single QUIC packet can grow to accommodate several ACK Ranges. Receivers need to limit the total number of ACK Ranges in a packet to prevent ACKs from becoming too large. The value of the limit is an implementation decision. It is normal for QUIC clients to occasionally send additional frames (e.g., a Ping frame) to force an ACK from a server.[1] For example, Quicly bundles a Ping frame with the next ACK when the number of ACK Ranges exceeds eight and allows a maximum of sixteen ranges in a packet to limit the total ACK size. The ACK size in Quicly stays comparable with TCP, between 60 and 90 bytes, with 1% packet loss on the forward path.

The ACK Ranges typically account for a small proportion of the size of a QUIC ACK packet. Figure 3 shows the breakdown of QUIC client traffic, as a percentage of the total traffic at the sender, for Firefox and Chromium. The figure combines results for all tested paths. Firefox seeks to use AR 2, resulting in a corresponding ACK volume between 2% and 3% of the total traffic. This is almost twice the value for TCP of 1.33% without loss (assuming TCP AR 2 and without including HTTP requests or TLS overhead). In contrast, Chromium uses AR 10, resulting in under 1% of the total traffic. The top contributors by volume to the packet size are the IP, QUIC and UDP headers, alongside per-packet crypto overhead and Crypto frames. The per-packet authentication tags contribute 90% of the total crypto overhead.

Recent analyses show an increase in return traffic for consumer networks, attributed to video-conferencing, remote work and similar applications.[37] Header compression for the IP and UDP headers is usually utilised to reduce the capacity in satellite and cellular networks. Our analysis in Table 3 estimates the size of the compressed header using ROHC compression ratios. The VPN overhead described is consistent with a standard

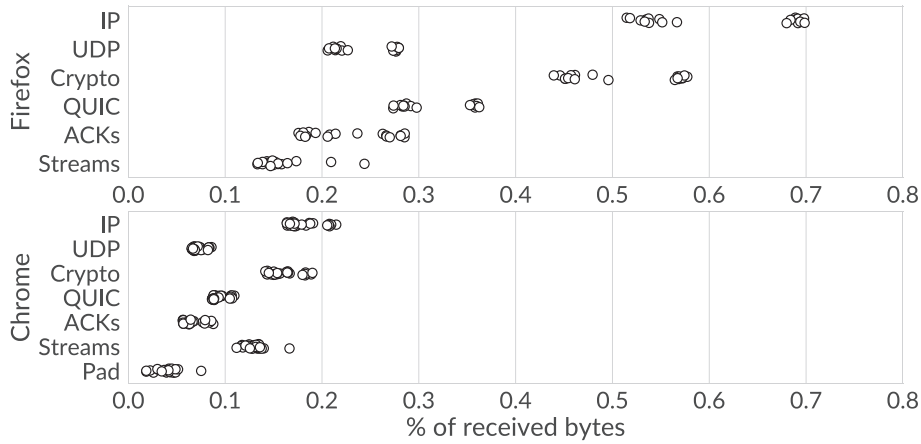


FIGURE 3 Breakdown by packet header and frames of the total number of bytes in the forward direction (represented as the proportion of the total bytes in the return direction) for Chrome 90 and Firefox 78 connections using HTTP/3 video. Each dot represents a measurement. This considers IP, UDP and QUIC headers, per-packet Crypto overhead, and Crypto, ACK, Stream and Padding frames

TABLE 3 Minimum ACK overhead per received packet (in bytes) for different ACK strategies, calculated considering the smallest size of ACKs (40 B for TCP and 51 B for QUIC). For VPNs and encrypted tunnels, QUIC reduces the end-to-end ACK rate. Conversely, TCP encapsulated by a VPN does not benefit from ACK Thinning or a split-TCP PEP on the path

	Default Size	TCP PEP	QUIC AR 10	ROHC for TCP AR 2 and QUIC AR 10
IPv4 TCP	20	4	N/A[2]	2
IPv6 TCP	30	6	N/A[2]	3
IPv4 QUIC	26	N/A[1]	5	03
IPv6 QUIC	36	N/A[1]	7	3
VPN v4 TCP	54	54	N/A[2]	43
VPN v4 QUIC	60	61	14	10

[1]Use of proxies to enhance QUIC remain an area of future research.
[2]TCP cannot use an updated AR policy.

configuration of OpenVPN (a popular VPN client), that uses UDP, AES-CBC and HMAC-SHA-256 (the VPN overhead varies drastically depending on the transport protocol, the cryptographic algorithm and the hash function). The size of an ACK is given for a received packet and assumes a connection in steady-state with no packet loss.

Table 3 shows that appreciable benefits can be achieved in QUIC using header compression, albeit not as large as in TCP. This benefit is greater when using the larger packet header of IPv6 (i.e., reducing a 71B QUIC ACK packet to 26B). When transport PDUs are further encapsulated, for example, when a client uses a VPN, link-layer header compression can only be performed on the outer header and methods such as ACK Thinning also cannot reduce the overhead. Similarly, a TCP split-PEP method does not provide benefit when forwarding VPN-encrypted packets. On the other hand, changing the AR is an end-to-end modification that reduces the number of bytes transmitted even when using encryption.[7] The table shows that for an IPv4 VPN, a change in the ACK policy for QUIC, for example, AR 10 can provide significant saving in capacity compared to TCP, and still further improvement is possible by additionally using ROHC.

5.3 | Bundling of QUIC frames

QUIC has the option to bundle multiple frames in the same packet. Regardless of the transport protocol used, client applications could need to send additional data when using HTTP/3. This subsection examines whether or not ACK frames were bundled by different clients when receiving video over HTTP/3.

Clients can send Stream frames for the lifetime of a connection (Figure 3). These include application requests, as well as cookies and telemetry. In these tests, we observed that Firefox and Chromium respectively opened an average of 11 and 14 connections for the duration of streaming 300 seconds of video. In this experiment, short connections (<400 packets) outnumber long-lived ones by 3–6 times.

While bundling can eliminate packet overhead, the amount of bundling varies by implementation. Chromium bundles ACK frames with other data or control frames in up to 20% of packets for long video streams. This effectively shares the cost of the IP, QUIC and Crypto headers between frames, eliminating overhead compared to sending these separately. For short streams, bundling was observed for both clients and can reach 100%, because in this case, a typical client sends more data than it receives. Bundling was not found to be currently prevalent in long streams when using Firefox. These results were not impacted by the path characteristics (such as capacity and RTT), but by the QUIC implementation decisions.

5.4 | Estimation of client traffic

To quantify the benefits of ACK reduction this section derives an analytical expression to estimate the volume in bytes of the ACKs generated by a client. The derived formula highlights the role of ACKs due to timeouts. In our experiments, we found that aside from ACK frames, only the initial cryptographic data exchange made a significant contribution. Other QUIC frames, such as the frames used for flow-control HTTP requests, generate only a small overhead (less than 5%).

Indicating the volume in bytes for all client ACKs as V_{bm} and the bytes in all TLS handshake packets sent by the client as V_{crypt}, the total traffic generated in one connection (T_{client}) can be approximated by:

$$T_{client} = V_{bm} + V_{crypt} \quad (1)$$

Denoted by A_{total}, the total number of ACKs generated upon receiving packets in-sequence, A_{loss} the number of ACKs generated after loss and S_{bm} the size of an ACK (which depends on implementation—see Table 2), the client traffic estimation becomes:

$$T_{client} = (A_{total} + A_{loss}) \cdot S_{bm} + V_{crypt} \tag{2}$$

The total number of sent ACK frames A_{total} depends on a combination of the AR, the value of the *ACK Delay* timer and the RTT. Similarly, A_{loss} depends on the policy used to generate ACKs when there is loss or reordering. A_{total} can be further divided into ACKs generated after an *ACK Delay* timeout, (A_{del}), and $(A_{total} - A_{del})$ ACKs generated after receiving *AR* packets. When the receiver *ACK Delay* expires, on average, only a fraction α of a sequence of *AR* packets is acknowledged. Analysing the Quicly transfers with different ARs, we find, on average, that α is one-half of the sequence of *AR* packets (for example, for an *AR* of 20, on average only 10 packets are ACKed when the *ACK Delay* timer expires). We therefore consider α to be 0.5.

Since ACKs acknowledge the entire sequence of $Serv_p$ packets from the server. If no losses occur, we have

$$A_{total} = Serv_p / AR + (1 - \alpha) \cdot A_{del} \tag{3}$$

This shows how the *ACK Delay* impacts the number of ACKs sent. A decrease in the *ACK Delay* timeout can result in an increase in the total number of ACKs, and therefore the volume of traffic.

A policy could set two different ARs, depending on whether or not the sender is in slow-start. When a sequence of initial P_{init} packets from the server are acknowledged using an AR_{init} with a PN that is different to the next expected AR_{sub}, we can write the Equation (3) to account for each AR. Including the number of ACKs in each period and assuming the same α, this gives

$$A_{total} = \left(\frac{P_{init}}{AR_{init}} + \frac{Serv_p - P_{init}}{AR_{sub}} \right) + (1 - \alpha) \cdot A_{del} \tag{4}$$

Finally, when a receiver observes reordering of packets (due to either loss or reordering), the receiver acknowledges the following k packets. Indicating with L the number of loss/reordering event clusters (e.g, within in the same RTT), this can be used to estimate the number of additional ACKs as

$$A_{loss} = k \cdot L \tag{5}$$

Figure 4 reports the T_{client} normalised to the traffic generated by the server, that is, T_{client}/T_{server} expressed as a percentage. AR 2 (the default by Quicly and Firefox) results in ACK volumes corresponding to 3% of server traffic.The figure also shows the median estimated T_{client} as

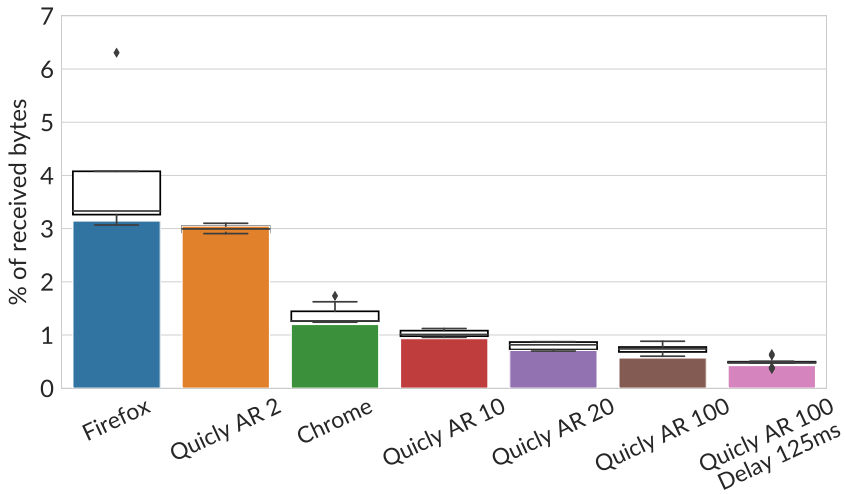


FIGURE 4 Client traffic as a percentage of the total bytes received from the server. Quicly was modified to use various ACK policies. This figure compares experimental data from measurement with the traffic estimated by Equation (3) in Section 5.4. The box plots show results for a 1 MB transfer over a 10 Mbps satellite path, while the vertical bars represent the median client traffic estimated by Equation (3) for the same data

calculated by Equation (3) for the same data. This estimate does not account for data contained within QUIC Stream frames, such as HTTP/3 requests or client cookies. These depend on the server and are not protocol-dependent, but could result in a more conservative estimate of the total volume of traffic sent by the client. The predicted number of received bytes for Firefox and Chrome in Figure 4 is accurate with a difference between the median value and the estimation of 0.19% and 0.07%, respectively. The volume predictions in Figure 4 are also accurate for Quicly, which does not embed an HTTP/3 server.

The interaction of the default *ACK Delay* (d = 25 ms) with the path RTT (>600 ms) can be observed when the AR is much greater than 10. The ACKs generated for AR 100 are sent every 25 ms due to *ACK Delay* instead of every 100 packets, resulting in an effective AR of 40, and a volume percentage of 0.73. To explore further, we increased the *ACK Delay* to approximately RTT/4 (d=125 ms) and observed the ACK traffic volume reduced to the estimated 0.25%. The data and estimates both show that additional benefit diminishes when reducing the AR beyond 10, both in bytes and packet overhead.

5.5 | AR interactions for a path with no packet loss

This section discusses the impact of an ACK policy for a scenario with no packet loss. An AR of N introduces a 'byte delay'[32] during slow-start equal to the time to send N packets back to back, or $(N-1)$ times the transmission time of the burst per RTT. This effect is cumulative and persists until the end of slow-start, as illustrated in Figure 7A. The authors previously demonstrated the byte delay effect on a simulated network using a version of Quicly without pacing,[7] shown in Figure 5.

The vertical axis shows the size of the *cwnd* at the sender. Each vertical line corresponds to an increase in *cwnd* after receiving acknowledgements for one round of packets. The gradient of the increase is determined by the rate of data packet transmission. In this case, the forward path was a 10 Mbit/s share of a multiplexed satellite link, with a shared return link. A higher rate forward path would not result in a different pattern of increase, but in a more vertical gradient. Therefore, the byte delay effect is only affected by a larger forward path capacity when the *cwnd* grows to exceed the Bandwidth-Delay Product (BDP).

On the horizontal axis, the time between each round of increase represents one RTT plus all transmission delays (the time to send the data, receive and process the acknowledgements, i.e., the byte delay). To explore the byte delay effect in non-simulated networks we measured the time to complete a 1 MB transfer. This was repeated 50 times using the networks listed in Table 1. The results are shown in the first three rows of Table 4. Regardless of RTT, a higher AR has little impact on the total time to download: in the case of 4G and Satellite, transfers using AR 10 complete on average approximately 1/2 RTT faster that AR 2 (50 and 400 ms, respectively), and for DSL approximately 1 RTT (40 ms) slower. The byte transmission delay is mitigated as the packets become dispersed across the RTT, by multiplexing with other traffic. Pacing mitigates micro-bursts resulting from larger ARs, but also introduces a delay in opening the *cwnd*. Using a reduced AR during slow-start (i.e., DAASS) can help reduce the effects of byte delay[10] and removes the reliance on pacing, at the expense of sending more bytes back to the sender. This is illustrated in Figure 4: Chromium has an extra 40 ACK packets overhead compared to Quicly because it uses an AR of 2 for the first 100 packets. The figure shows the measured data is in agreement with Equation (3) described in Section 5.4.

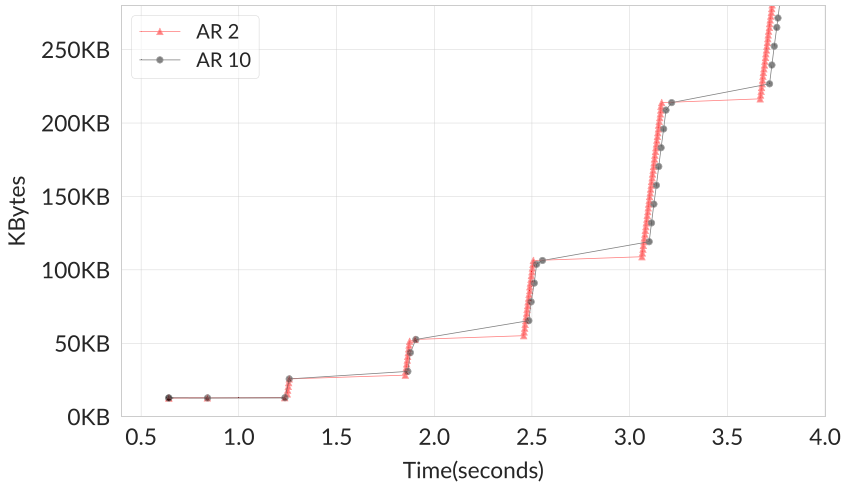


FIGURE 5 The byte delay can increase cumulatively with each RTT between AR 2 and AR 10. This was previously measured in a simulated network with a Quicly version that did not support pacing. The *cwnd* growth is not otherwise affected by the AR difference

TABLE 4 Median time to download a 1 MB file, in seconds, with different ARs and ACK policies after reordering. In the case of the satellite, larger ARs correspond to an increase in throughput

	4G	DSL	Satellite
AR 2	0.72	0.44	5.74
AR 10	0.69	0.45	5.64
AR 20	0.67	0.48	5.33
AR 2 1% Loss D	2.18	1.14	20.59
AR 10 1%Loss D	1.88	1.01	18.83
AR 20 1%Loss D	2.11	1.64	16.54
AR 2 1% Loss NR	1.81	1.14	18.63
AR 10 1%Loss NR	2.65	1.25	19.88
AR 20 1%Loss NR	2.56	1.39	17.10
AR 2 1% Loss 3A	2.01	1.25	18.35
AR 10 1%Loss 3A	2.30	1.24	16.82
AR 20 1%Loss 3A	2.30	1.71	17.72

Note: $N = 50$ measurements.

5.6 | AR interactions for a path with packet loss

This section discusses the impact of an ACK policy when there is loss. A QUIC receiver sends an ACK immediately when an out-of-order packet is detected, resetting the AR counter and *ACK Delay* timer.[1]

A loss is detected when the PN difference between the two most recent packets is greater than the reordering threshold at the sender. The recommended reordering threshold is 3 packets (to match TCP), which means a loss will only be detected after the second ACK following a reordering event. A larger AR can delay loss detection, depending on a sender's sensitivity to reordering. This delay is illustrated in Figure 7B.

To explore the delay, we repeated the 1 MB transfers 50 times adding a 1% random symmetric (affecting both server and client) packet loss. This loss percentage was chosen to compare our results with Volodina et al.[34] We expected to see the effects of this delay in the time to complete a 1 MB transfer, and for it to be inversely proportional to the RTT of the connection. In the case of RTTs under 20 ms where pacing is not used, this pathology is known to exist.[34] We did not consistently observe this effect in any non-simulated network. This can be explained by the presence of cross-traffic, QUIC implementation pacing and RTT duration. Table 4 presents the time (seconds) to download a 1 MB file for various ARs (2, 10 and 20) and using three different strategies to detect reordering: the default behaviour (D) where a receiver sends an ACK when reordering is detected, no reordering detection (NR) where a receiver does not send an ACK immediately upon detecting reordering, and reverting to an AR of 1 for the next $k = 3$ packets upon detecting reordering (3A).

Table 4 investigates the extent to which the default behaviour mitigates loss detection delay. We did not find consistent throughput changes for different reordering detection methods, even when increasing the packet loss to 5%. This suggests that QUIC performance is resilient to delays in ACK responses. The 3A strategy shows an interesting trade-off, because the additional ACKs do not contribute significantly to the volume of data sent (for a 1% packet loss, this is 1% of the total number of bytes in the forward direction), but do add robustness against spurious ACK reordering from particular patterns of loss and/or traffic.

Figure 6 shows the distribution of time to download 1 MB for no loss and 1% random sender loss with default reordering detection for the satellite path. When using AR>2, the *ACK Delay* choice with respect to the RTT impacts performance in the case of loss. ACK Delay timeouts could occur in the case where throughput is limited by losses. The timeout expires before the receiver has received enough packets to emit an ACK.

Table 5 showcases how frequent the *ACK Delay* timeout occurs in the case of loss and no loss, for paths with different RTTs and different ARs. In the case of DSL, with AR 10 and AR 20, 10% and 25% respectively of all ACKs are generated due to a timeout—this RTT is approximately the *ACK Delay*. Where the *ACK Delay* value is at least four times smaller than the RTT (in the case of 4G), this can account for a half of all generated ACKs. For an RTT much higher than the *ACK Delay*, this can rise to 80%. Table 5 also shows timeouts when there was no loss, increasing with the AR and where the RTT is much larger than the *ACK Delay*. These timeouts can occur during slow-start when *cwnd* is limited. This case is similar to that of throughput limited by losses.

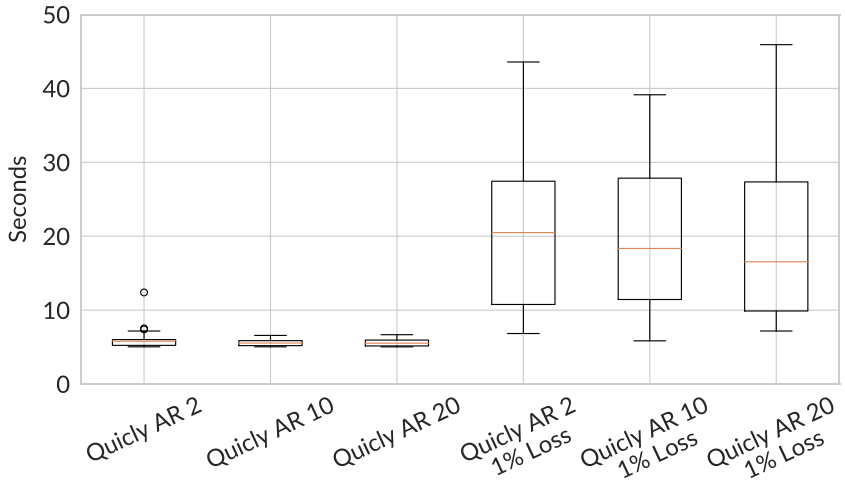


FIGURE 6 Time to download 1 MB, for different ARs, using a satellite link with no loss and with emulated 1% loss

TABLE 5 Percentage of time (median) an ACK is generated due to an *ACK Delay* timeout, for different ARs and network paths

	4G	DSL	Satellite
AR 2 No Loss	2%	1%	2%
AR 10 No Loss	6%	3%	32%
AR 20 No Loss	8%	7%	30%
AR 2 1% Loss	5%	2%	33%
AR 10 1%Loss	19%	10%	78%
AR 20 1%Loss	45%	25%	79%

Note: $N = 50$ measurements.

5.7 | Discussion of results

In July 2022, the AF Frame draft specification[6] was updated to recommend a minimum of one ACK per RTT, as well as pacing to mitigate the effects of infrequent ACKs. Sending an ACK twice per received window appears sufficient to allow full utilisation of the pipe for a single flow. In general, an *ACK Delay* that is smaller than RTT/4 can result in an increased number of ACKs due to repeated timer firing (Table 5). For a small *cwnd*, a smaller *ACK Delay* can prevent unnecessary feedback delay and avoid throughput reduction.[34] We suggest doubling this to 4 to ensure frequent feedback that is robust to isolated ACK packet loss. The *ACK Delay* can be changed using the AF Frame after the sender has computed the RTT.

We suggest this policy alone may not be robust, because it is not responsive for long RTTs, and clients would rely on senders to calculate the RTT. To accommodate paths with a *cwnd*>40 packets (e.g., a GEO satellite system), we suggest a receiver policy where a receiver sends an ACK frame for at least every *N* received packets (AR N). This will mitigate the side effects of pacing, and also can adapt faster where there is a significant change in the path delay requiring the sender to re-compute the RTT.

There are trade-offs in the choice of AR: Section 5.5, shows that the choice of AR introduces two types of delay: byte delay and loss detection delay. Figure 4 shows that using an AR greater than 10 does not provide compelling benefits in terms of byte reduction, and can introduce other issues: if an ACK is lost for AR of *N*, a sender would then receive an ACK covering 2 * *N* segments. This means that a sender can release a significant number of packets into the network and so a sender needs to rely on pacing to mitigate these bursts. However, pacing also delays the reception of ACKs, reducing the throughput (Figure 7A). The size of this penalty depends on the design of the pacing algorithm, and the delay of the path. Although pacing is a key part of the QUIC specification, no specific pacing algorithm is defined.[1]

We consider AR 10 to be safe, based on other observations that routers can buffer a burst of this size (the value 10 is widely used as the TCP initial *cwnd*). We found several current implementations (see Section 5.1) have adopted this default AR of 10, as well as CC mechanisms other than Reno. Furthermore, results in Sections 5.2 and 5.4 show significant byte and packet reduction when using AR 10 compared to QUIC's default AR of 2. To avoid byte delay for any CC implementing slow-start, senders could start with AR 2 and later transition to using an AR of 10 after a

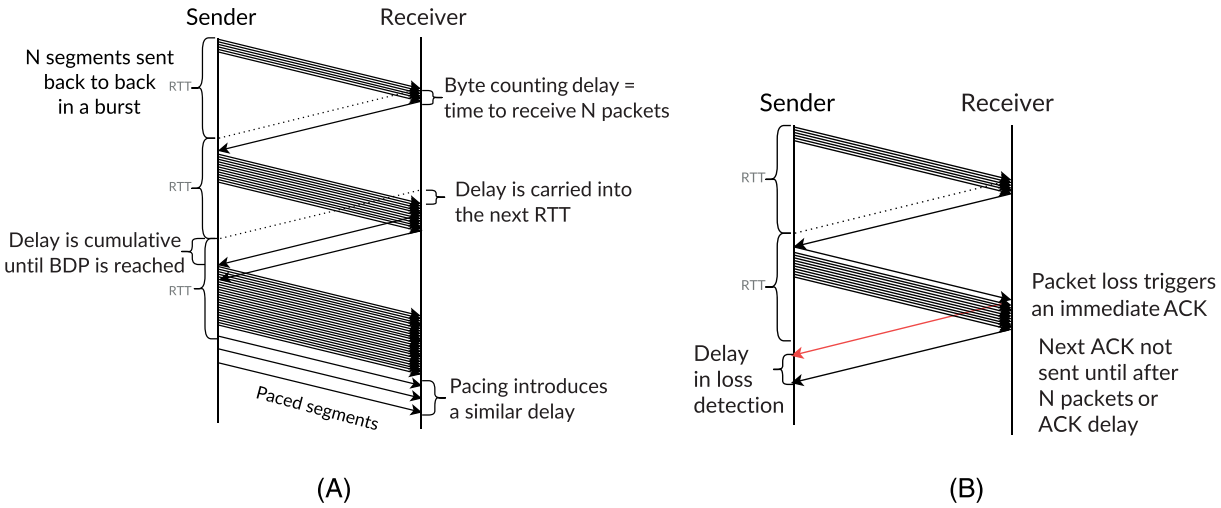


FIGURE 7 Byte counting delay for AR *N* (A) and Loss delay for AR *N* (B). Pacing delay has a similar effect and compounds in the case of loss. The red (top) arrow in sub-figure (B) shows an ACK sent as a result of reordering detection

small number of RTTs, or a set number of packets. Bundling ACK frames with other data packets was not prevalent in all the implementations studied (see Section 5.3), although it was seen to reduce overhead.

Loss detection delay can be mitigated by increasing the ACK frequency immediately after detecting reordering.[6] Our results in Section 5.6 show that sending an ACK immediately when reordering is detected does not significantly increase the overhead. An earlier draft for QUIC previously recommended reverting to AR 1 for a quarter of an RTT, but this was removed after consensus within the IETF working group that this would generate too many ACKs. We observed no appreciable performance impact from sending a total of $r = reordering\_threshold$ ACKs after reordering is detected.

6 | RECOMMENDATIONS AND IMPLICATIONS

The previous sections have presented experimental results that evaluate a set of policies in cellular, terrestrial and satellite networks and show that throughput can be maintained even when a receiver sends fewer acknowledgements. We analysed the factors that influence a suitable ACK policy and show that QUIC's standard-specified default value for the *ACK Delay* period can lead to additional delays on paths with short RTTs.

We have also shown that AR 10 is suitable for all paths we considered and recommend this value for all Internet paths, to lower transport processing and transmission costs and reduce return path loading. We also recommend sending one ACK frame at least every RTT/4 (see Section 5.6), based on the observed path RTT, where this is controlled by the sender.[6] We recommend these two methods for ACK generation are used together. For CCs implementing slow-start, sending an ACK every 2 received packets during slow-start can help prevent throughput loss; we recommend this for the first 100 packets of a transfer, because the additional load presented by ACKs is not significant until the *cwnd* grows.

We also recommend using QUIC frame bundling to reduce overhead associated with ACKs, and enabling UDP header compression to improve performance for paths with limited capacity. QUIC is expected to continue to evolve, and some developments (such as ACK frame padding as a mitigation for traffic analysis) could also increase the ACK volume. Such methods would be undesirable for the paths considered in this paper. If this emerges as a common method, research will be needed to explore this area.

6.1 | Implications for Internet service providers

Optimisations over the years have resulted in TCP delivering ACKs that cover several packets. Moreover, TCP ACK Thinning and split-TCP PEPs have been deployed in networks. Because these methods only help unencrypted traffic, the rise in popularity of Virtual Private Networks (VPN)s[38] causes a significant concern. Table 3 tabulates the return traffic volume per received packet and shows the current QUIC specification results in increased volume compared to TCP. Compared to TCP, we have also shown that QUIC ACKs are larger. However, changing the AR in QUIC has benefit whether or not the traffic is further encapsulated.

A QUIC sender also sends several other control frames (Figure 3). The main contributor to QUIC ACK volume in bytes is the IP header of packets, which is compressible. There is benefit in enabling IP and UDP header compression for QUIC packets when optimisations are required in the return path. This benefit is larger for IPv6.

The network operations community has also long relied on being able to understand Internet traffic patterns, both in aggregate and at the flow level, to support network management, traffic engineering and troubleshooting.[12] Widespread deployment of transport protocols such as QUIC will impact network operations practice, requiring research into alternative methods that work without access to the transport header.[12]

6.2 | Implications for QUIC stack implementers

Standards development continues and will enable developers to implement critical performance-enhancing methods. The flexibility of QUIC enables a more rapid development of new transport methods, including developments that enable an ACK policy to reduce the ACK Frequency. A question remains about whether clients using all Internet paths would benefit from less frequent acknowledgements. We recommend implementers follow the guidelines presented at the beginning of this section. A change from the default AR can be implemented using the AF Frame, and this can also be used to update it during the lifetime of a connection.

Once a receiver can use a larger AR, information at the sender could in future be used to further tune the AR noting that ACKs are also used to detect loss and congestion (e.g., tuning that reflects server understanding of the path RTT, choice of CC and traffic profile).

6.3 | Implications for Internet content providers

The design of the QUIC protocol changes the place where performance enhancement needs to be provided. Changing the QUIC ACK frequency has benefit without the need for an additional in-network mechanism. This can also be used even when QUIC is itself carried over a VPN, a significant performance advantage compared to TCP. The ACK Frequency frame allows changing the *ACK Delay* and the AR for a connection 3. Since content providers can control the settings of deployed QUIC servers, they can take advantage of changing the ACK policy by relating the settings to measured path characteristics.

6.4 | Implications for systems design

Previous sections have discussed design implications for QUIC senders and receivers. Performance evaluation has shown that QUIC/HTTP/3 generally can work well over a range of asymmetric radio technologies, including a GEO satellite path. This subsection looks at the implications for the design of equipment.

The pace of the development and rapid deployment of QUIC is also having a disruptive impact on the way communications systems are designed and used to support Internet applications. PEPs have been widely used in communications systems, and a variety of PEP mechanisms have been deployed, both to accelerate TCP performance for users and to optimise use of capacity, especially return link capacity. These methods are proven and will continue to be needed for TCP for high RTT paths, but traditional PEPs do not accelerate QUIC or traffic using VPN Encryption. Designs that support IP/UDP compression will continue to see benefit with UDP traffic, including QUIC. The Internet share of QUIC traffic is only likely to increase for all networks, therefore equipment designs need to be ready for the changes that come with QUIC.

This paper has shown that QUIC can be updated to minimise return capacity usage, and this paper and previous work have shown this can significantly reduce constraints due to path asymmetry. Future research is expected to identify other ways in which improvement in performance could be achieved (e.g., to enable new CC methods or improvements to flow control for QUIC). It is important to note that these new mechanisms will need to be taken-up by QUIC implementers, and deployed in the Internet (e.g., at CDN nodes and web browser clients), rather than by changing communications equipment such as the satellite ground segment.

7 | CONCLUSIONS

This paper explored the use of the IETF QUIC transport protocol with a focus on the implications of a receiver using different ACK policies. In the past, most Internet traffic used TCP. PEPs could be used to increase the throughput and provide trade-offs that reduce consumption of capacity (e.g., reducing the number of packets, or eliminating unnecessary protocol header information). These long-standing techniques no longer provide benefit, because of the use of transport encryption. This means that system designers and network operators no longer control protocol trade-offs. New approaches are therefore needed to accelerate the performance.

The paper analysed the performance of the latest specification of the QUIC protocol over a range of network paths, with an emphasis on the return path traffic. It then explored the interaction with a range of network segments, including satellite systems. It evaluates design choices for how a QUIC receiver generates acknowledgements and it then proposes a recommendation for all Internet paths, based on evaluation of a range of path characteristics and ACK policies. An ACK policy has been designed that reduces the volume/rate of traffic sent on the return path and can reduce the associated processing costs in endpoints, without sacrificing throughput. Where the Internet path presents a high asymmetry this policy has been shown to improve throughput. In contrast to previous methods that change network operator equipment (such as satellite ground segment equipment), this policy updates QUIC implementations by extending the QUIC protocol.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in ijscn-reducing-ack-frequency-captures at https://github.com/uoaerg/-ijscn-reducing-ack-frequency-captures.

ORCID

Ana Custura https://orcid.org/0000-0002-8627-2212

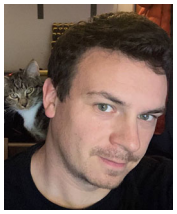
REFERENCES

1. Iyengar J, Thomson M. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, IETF; 2021.
2. Cui Y, Li T, Liu C, Wang X, Kühlewind M. Innovating Transport with QUIC: Design Approaches and Research Challenges. *IEEE Int Comput*. 2017;21(2):72-76. https://doi.org/10.1109/MIC.2017.44
3. Li T, Zheng K, Jadhav R, Kang J. Optimizing ACK mechanism for QUIC. IETF Work in Progress. 2022. https://datatracker.ietf.org/doc/draft-li-quic-optimizing-ack-in-wlan/04/
4. Fairhurst G, Yun A. Design of the DVB-RCS2 higher layer satellite architecture. *Int J Satellite Commun Netw*. 2013;31(5):233-248.
5. Oku K, Iyengar J. Can QUIC match TCP's computational efficiency? Online; Accessed on 2022-07-21. 2020. https://www.fastly.com/blog/measuringquic-vs-tcp-computational-efficiency
6. Iyengar J, Swett I. QUIC Acknowledgement Frequency. IETF Work in Progress. https://datatracker.ietf.org/doc/draft-ietf-quic-ack-frequency/02/; 2022.
7. Custura A, Jones T, Fairhurst G. Impact of Acknowledgements using IETF QUIC on Satellite Performance. In: 10th Advanced Satellite Multimedia Systems Conference. IEEE; 2020:1-8.
8. Rüth J, Kunze I, Hohlfeld O. TCP's Initial Window–Deployment in the Wild and Its Impact on Performance. *IEEE Trans Netw Serv Manag*ement. 2019;16(2):389-402. https://doi.org/10.1109/TNSM.2019.2896335
9. Yu L, Minhua Y, Huimin Z. The improvement of TCP performance in bandwidth asymmetric network. In: 14th Personal, Indoor and Mobile Radio Communications Conference, Vol. 1. IEEE; 2003:482-486.
10. Custura A, Jones T, Fairhurst G. Rethinking ACKs at the Transport Layer. In: FIT Workshop, Networking Conference. IFIP/IEEE; 2020:731-736.
11. Bishop M. HTTP/3. RFC 9114, IETF; 2022.
12. Fairhurst G, Perkins C. Considerations around Transport Header Confidentiality, Network Operations, and the Evolution of Internet Transport Protocols. RFC 9065, IETF; 2021.
13. Balakrishnan H, Padmanabhan V, Fairhurst G, Sooriyabandara M. TCP Performance Implications of Network Path Asymmetry. RFC 3449, IETF; 2002.
14. Chen J, Gerla M, Lee YZ, Sanadidi MY. TCP with delayed ACK for wireless networks. *Elsevier Ad Hoc Netw*. 2008;6(7):1098-1116.
15. Bormann C, Burmeister C, Degermark M, et al. RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed. RFC 3095, IETF; 2001.
16. Engan M, Casner S, Bormann C. IP Header Compression over PPP. RFC 2509, IETF; 1999.
17. Rawat P, Bonnin JM, Toutain L, Choi Y. Robust Header Compression Over Long Delay Links. In: Vehicular Technology Conference. IEEE; 2008:2136-2141.
18. Kojo M, Griner J, Shelby Z. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, IETF; 2001.
19. Caviglione L, Celandroni N, Collina M, et al. A deep analysis on future web technologies and protocols over broadband GEO satellite networks. *Int J Satellite Commun Netw*. 2015;33(5):451-472.
20. Cardaci A, Caviglione L, Ferro E, Gotta A. Using SPDY to improve Web 2.0 over satellite links. *Int J Satellite Commun Netw*. 2017;35(4):307-321.
21. Caini C, Firrincieli R, Lacamera D. PEPsal: A Performance Enhancing Proxy for TCP satellite connections. *IEEE Aerosp Electronic Sys Mag*. 2007;22(8):7-16.
22. Chan M, Cheriton D. Improving Server Application Performance via Pure TCP ACK Receive Optimization. In: Annual Technical Conference. USENIX Association; 2013:359-364.
23. Cai Q, Chaudhary S, Vuppalapati M, Hwang J, Agarwal R. Understanding Host Network Stack Overheads. In: SIGCOMM 2021 Conference. ACM; 2021; New York, NY, USA:65-77.
24. Allman M, Paxson V, Blanton E. TCP Congestion Control. RFC 5681, IETF; 2009.
25. Deland-Han. New registry entry for controlling TCP ACK - Windows Server. Online;. Accessed on 2022-06-01. https://docs.microsoft.com/en-us/troubleshoot/windows-server/networking/registry-entry-control-tcp-acknowledgment-behavior
26. Allman M. On the generation and use of TCP acknowledgments. *ACM CCR*. 1998;28(5):4-21.
27. Cardwell N, Cheng Y, Yeganeh SH, Swett I. Van Jacobson BBR Congestion Control. IETF Work in Progress. 2022. https://datatracker.ietf.org/doc/draft-cardwell-iccrg-bbr-congestion-control/02/
28. Tran VH, Bonaventure O. Beyond socket options: Making the linux TCP stack truly extensible. In: Networking Conference. IFIP/IEEE; 2019:1-9.
29. Yang P. Linux Kernel: Merge branch: fix stretch ACK bugs in congestion control modules. Online; 2020. https://lkml.org/lkml/2020/3/16/54

30. Marx R, Herbots J, Lamotte W, Quax P. Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity. In: Workshop on the Evolution, Performance, and Interoperability of QUIC. ACM; 2020; New York, USA:14-20.
31. Yang X, Eggert L, Ott J, Uhlig S, Sun Z, Antichi G. Making QUIC Quicker With NIC Offload. In: Workshop on the Evolution, Performance, and Interoperability of QUIC. ACM; 2020; New York, USA:21-27.
32. Landström S, Larzon LA. Reducing the TCP Acknowledgment Frequency. *ACM CCR*. 2007;37(3):5-16. https://doi.org/10.1145/1273445.1273447
33. Völker T, Volodina E, Tüxen M, Rathgeb EP. A QUIC Simulation Model for INET and its Application to the Acknowledgment Ratio Issue. In: Networking Conference. IFIP/IEEE; 2020:737-742.
34. Volodina E, Rathgeb E. Impact of ACK Scaling Policies on QUIC Performance. In: 46th Conference on Local Computer Networks IEEE; 2021:41-48.
35. Iyengar J, Swett I. QUIC Loss Detection and Congestion Control. RFC 9002, IETF; 2021.
36. Seemann M. QUIC Interop Runner. Online; 2021. Accessed on 2022-05-21. https://interop.seemann.io
37. Arkko J. Report from the IAB COVID-19 Network Impacts Workshop 2020. RFC 9075, IETF; 2021.
38. Longworth J. VPN: From an obscure network to a widespread solution. *Comput Fraud Security*. 2018;2018(4):14-15.

AUTHOR BIOGRAPHIES

Ana Custura is currently undertaking a PhD in Internet Engineering at the University of Aberdeen. She has been active in the field of Internet measurement for the past 5 years, and her publications have helped shape several IETF engineering standards. Her interests include network measurement and transparency, privacy and digital freedom, amateur radio and electronics. She is also a fan of Free Software and in her spare time maintains several tools for network measurement in Debian and Tor Project.



Tom Jones is an Internet Engineer working in the IETF and on practical implementation of new ideas in Operating Systems. His research interests includes advancement in the programming API to the network, scalability and performance of new protocols and the integration of new protocols in satellite environments. He is an active contributor to the IETF and has published RFCs on the UDP API and new methods for Path MTU Discovery.

Dr. Raffaello Secchi is a Lecturer at the University of Aberdeen since 2016. His research has combined simulation, analysis and performance measurement to focus on Internet transport and protocols. His work contributions to developing scalable and adaptive Internet congestion control and new transport mechanisms for IP-based satellite communications include journal publications and participation in Internet standards. He actively participated in several EU research projects (RITE, MONROE, and SMILE) and is currently involved in the ESA QUICOPTSAT to integrate new QUIC secure connections into the satellite ecosystem.

Dr. Gorry Fairhurst is a Professor in Internet Engineering. His research interests include protocol mechanisms, wide scale Internet measurement, IP multicast services, and satellite systems. He contributed to ETSI to standardise broadband satellite networking architectures and to the DVB-project on the GSE link encapsulation and IP-based TDMA satellite system architectures, and worked with ESA on a range of transport-related topics for broadband satellite. He is an active participant in engineering Internet standards at the IETF and has developed over 30 standards documents. His contributions include path MTU discovery, TCP, differentiated services, the QUIC transport, and Internet congestion control. He currently chairs the IETF Transport and Services Working Group.

How to cite this article: Custura A, Jones T, Secchi R, Fairhurst G. Reducing the acknowledgement frequency in IETF QUIC. *Int J Satell Commun Network*. 2022;1-16. doi:10.1002/sat.1466