EDITOR: Yong Cui, cuiyong@tsinghua.edu.cn

Acknowledgment on Demand for Transport Control

Tong Li and Kai Zheng, *Huawei, Shenzhen, 518129, China*
Ke Xu, *Tsinghua University & BNRist, Beijing, 100084, China, and PCL, Shenzhen 518066, China*

The dependence on frequent acknowledgments (ACKs) is an artifact of current transport protocol designs rather than a fundamental requirement. Frequent ACKs waste resources when the overhead of ACKs is nonnegligible. However, reducing the number of ACKs may hurt transport performance. "Tame ACK" is an on-demand ACK mechanism that seeks to minimize ACK frequency, which is exactly what is required by transport.

Data delivery between endpoints depends on backward signaling, such as acknowledgments (ACKs) for transport control. ACK mechanism is a common component for transport protocols such as Transmission Control Protocol (TCP) and Quick UDP Internet Connections (QUIC).[6] ACKs may serve multiple purposes. They acknowledge received packets, drive loss detection, trigger transmission of new packets, and assist in sending rate determination and round-trip timing.

Ideally, an ACK should be generated each time a packet is received at the destination. However, frequent ACKs introduce overhead for both the network and the end hosts. For example, in wireless networks running over the IEEE 802.11 medium access control protocol, ACK packets cause similar medium access overhead as data packets despite the much smaller size of the ACKs. Sending frequent ACKs preempt resources that could have been used for data packet transmission. Second, in asymmetric networks (e.g., cellular networks) with a much lower capacity on the uplink than the downlink, the increased upload traffic caused by data-intensive applications (e.g., live webcasting) can lead to congestion on the ACK path of the download traffic. Furthermore, sending frequent ACKs also induces computational cost at all nodes along the end-to-end path. Consequently, reducing ACK frequency can potentially improve transmission performance when the overhead of ACKs is nonnegligible.

However, over the past four decades, only few modifications to ACK mechanism have been deployed. One primary reason is that current protocol design couples the high ACK frequency with transport controls, such as robust loss recovery, accurate round-trip time (RTT) monitoring, and effective send rate control. For example, loss detection might be delayed when the number of ACKs is excessively reduced, sending fewer ACKs proportionally reduces the number of RTT samples that are capable to be generated by the sender, and acknowledging a large amount of data at once might lead to traffic burst. Multiple-component interactions make these challenges harder to be addressed at the same time.

RFCs 1122 and 5681 were two core functionality standards that introduced, the delayed ACK mechanism, the default ACK mechanism in most Linux distributions. It increases the number of incoming packets (counted before sending an ACK) from one to two and also sets a timer that avoids an ACK being excessively delayed. However, the delayed ACK mechanism is far from being optimal in terms of ACK frequency minimization. RFCs 4341 and 5690 described an ACK congestion control mechanism in which the minimum ACK frequency allowed is twice per send window. RFC 3449 discussed the imperfection and variability of TCP's ACK mechanism because of asymmetric effects and recommended reducing ACK frequency as a mitigation to these effects. However, these algorithms only solve part of the problems. For example, they did not address the interference on the RTT monitoring caused by reducing the ACK frequency. In the work by Li *et al.*[11] we demonstrated the

1089-7801 © 2020 IEEE
Digital Object Identifier 10.1109/MIC.2020.3045208
Date of current version 16 April 2021.

possibilities of simultaneously addressing all the challenges mentioned earlier by decoupling high ACK frequency from transport requirement.

Another reason is that ACK mechanism significantly impacts the logic of ACK-clocking algorithms (e.g., loss recovery, RTT monitoring, and send rate control). The robustness of these changes relies heavily on extensive unit and end-to-end testing, which were limited by the kernel APIs and memory-constrained kernel space. A major change of the whole protocol was difficult to be updated and deployed. Fortunately, the user-space developments of transport protocols (e.g., QUIC) allow for extensive logging and debugging, the high deployment velocity allows deploying ACK mechanisms with major modifications to the whole protocol stack.[2]

Based on these observations, it is time to rethink the current trajectory of ACK mechanism for transport control. Some work-in-progress drafts have paid great attention to ACK scaling technologies in the IETF working groups. For example, Fairhurst *et al.*[3] recommended reducing the ACK frequency by sending an ACK for at least every 10 received packets, and Kuhn *et al.*[7] recommended an ACK frequency of 4 ACKs every RTT, aiming to reduce link transmission costs for asymmetric paths. Instead of using an empirical value of ACK frequency, Iyengar *et al.*[5] recommended an extension of sender controlled ACK-FREQUENCY frame to make it tunable for the frequency of the delayed ACK mechanism.

An ACK mechanism can be called "on-demand" if the ACK frequency is minimized but the ACKs sent are exactly required by the transport. In contrast to the delayed ACK mechanism that adopts the maximum value between *byte-counting* and a timer, the authors of this article adopt the minimum one and propose Tame ACK (TACK),[9,10] an on-demand ACK mechanism that seeks to minimize ACK frequency with corresponding improvements in transport control to compensate for sending much fewer ACKs.

CHALLENGES FOR REDUCING ACK FREQUENCY

Before diving into the TACK design, we first discuss the possible challenges that need to be overcome when an ACK mechanism reduces ACK frequency. Specifically, as shown in Figure 1, we focus on the interactions between ACK mechanism and other protocol components, such as loss recovery, RTT monitoring, and send rate control.

Loss Recovery

For ordered bytestream transport, when loss occurs and a packet has to be retransmitted, packets that have already arrived but that appear later in the

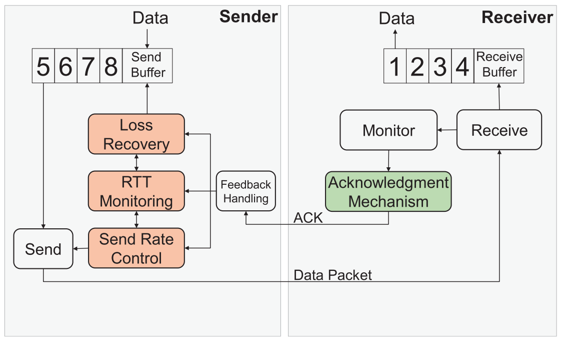


FIGURE 1. Interaction among protocols components.

bytestream must await delivery of the missing packet so the bytestream can be reassembled in order. Known as head-of-line blocking (HoLB), this incurs high delay of packet reassembling and, thus, can be detrimental to the transport performance.[12] Considering an ACK mechanism that sends much fewer number of ACKs than the legacy way, the delay incurred by HoLB will increase. If packet loss exists on the ACK path or the retransmission is lost again, then the delay doubles.

Landström *et al.*[8] tried to solve part of the problems by modifying the fast recovery scheme in this case. However, they did not consider the feedback robustness under excessive ACK losses. In the work by Li *et al.*,[11] we gave an example of a full framework of loss recovery that achieves both timely and robust feedback.

RTT Monitoring

The central premise of a closed control loop is that the RTT is well monitored. For example, the minimum RTT is used to update the TACK frequency. Many congestion controllers (e.g., BBR[1]) use the minimum RTT to estimate the congestion window. The minimum RTT is also used by QUIC loss detection to reject implausibly small RTT samples. However, the minimum RTT monitoring turns out to be one of the biggest victims of reducing ACK frequency.

The initial RTT can be computed during handshakes [see Figure 2(a)], after that, the sender calcu-

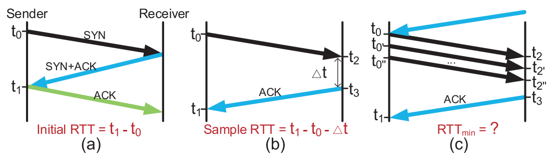


FIGURE 2. Case study of RTT monitoring. (a) Initial RTT computation. (b) ACK-clocking RTT sampling. (c) Minimum RTT monitoring when sending fewer ACKs.

lates an RTT sample upon receiving an ACK. For example in Figure 2(b), a packet is sent at time t_0 and arrives at time t_2. Assume that the ACK is generated and sent at time t_3, the receiver computes the ACK delay $\Delta t = t_3 - t_2$. The sender, therefore, computes the RTT according to Δt, t_0 and the ACK arrival time (t_1), i.e., $\mathrm{RTT} = t_1 - t_0 - \Delta t$.

However, as shown in Figure 2(c), when sending fewer ACKs, more data packets might be received during the ACK interval, generating only one RTT sample among multiple packets is likely to result in biases. For example, a larger minimum RTT estimate. In general, the higher the throughput, the larger the biases.

One alternative way to reduce biases can be that each ACK carries the per-packet timestamps for the sender to generate more RTT samples. However, 1) the overhead is high, which is unacceptable especially for high-bandwidth transport. Also, 2) the number of data packets might be far more than the maximum number of timestamps that an ACK is capable of carrying. In the work by Li *et al.*[11] we proposed a round-trip timing method based on the one-way delay (OWD) in order to reduce the biases introduced by sending fewer ACKs.

Send Rate Control

A burst of packets can be sent in response to a single delayed ACK. Legacy TCP usually sends microbursts of one to three packets, which is bounded according to definition of TCP's delayed ACK mechanism (RFCs 1122 and 5681). However, the fewer ACKs sent, the larger the bursts of packets released. When an ACK excessively delays ACKs, the burst send pattern is nonnegligible as it may have larger buffer requirement, higher loss rate, and longer queuing delay if not carefully handled.

To overcome the hurdles created by reducing ACK frequency, RFC 3465 described the appropriate byte counting (ABC) algorithm and limited the number of packets sent (i.e., two) in response to each incoming ACK to deal with feedback lags and traffic bursts. RFC 3449 also introduced some techniques (e.g., pacing) to handle the reduced ACK frequency to retain the TCP sender's ACK-triggered self-clocking. In the work by Li *et al.*,[11] we recommended pacing and demonstrated a co-designed BBR. Since BBR's RTT and bandwidth monitoring are coupled with frequent ACKs, we proposed a receiver-based paradigm that moves the monitoring logic from sender to receiver.

TACK: MINIMIZING ACK FREQUENCY

In this section, we give the frequency formula for TACK and its performance results.

Delayed ACK

By default, the current transport protocols, such as TCP and QUIC, specify a simple delayed ACK mechanism that a receiver can send an ACK for every other packet (i.e., byte-counting), or when the maximum ACK delay (α) timer expires. The frequency of the delayed ACK mechanism (f_{delayed}) is given as follow:

$$f_{\mathrm{delayed}} = \max\left\{\frac{\mathrm{BW}}{L \cdot \mathrm{MSS}}, \frac{1}{\alpha}\right\} \tag{1}$$

where L indicates the number of full-sized data packets counted before sending an ACK, α indicates the time interval between two ACKs, BW is the data throughput, and MSS is the maximum segment size.

In practice, the delayed ACK mechanism can only reduce limited number of ACKs. As described in RFC 1122 and updated in RFC 5681, L is strictly limited up to 2, and α is tens to hundreds of milliseconds and varies in different Linux distributions.

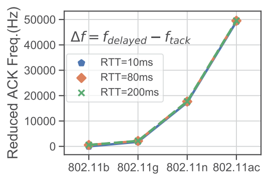
Even when L is allowed to be set larger than 2, the delayed ACK mechanism is far from being optimal. According to (1), when data throughput is high, ACK frequency is also high. When data throughput is low, ACK frequency is still as high as that in the case of high throughput, which wastes resources. This reveals that the frequency of the delayed ACK mechanism is not bounded or not minimized under bandwidth change.

Tame ACK (TACK)

TACK aims to minimize ACK frequency in the context of network dynamics. In contrast to the delayed ACK mechanism that adopts the maximum value between byte-counting and a timer, TACK adopts the minimum one. That is, $f_{\mathrm{tack}} = \min\{\frac{\mathrm{BW}}{L \cdot \mathrm{MSS}}, \frac{1}{\alpha}\}$. In practice, α can be set to a fraction of RTT (RFC 4341), i.e., $\alpha = \frac{\mathrm{RTT}_{\min}}{\beta}$. $\mathrm{RTT}_{\min}$ is the minimum RTT observed over a long period of time, and β indicates the number of ACKs per $\mathrm{RTT}_{\min}$. As a consequence, the frequency of TACK is in reality given as follow:

$$f_{\mathrm{tack}} = \min\left\{\frac{\mathrm{BW}}{L \cdot \mathrm{MSS}}, \frac{\beta}{\mathrm{RTT}_{\min}}\right\}. \tag{2}$$

The frequency of TACK is decided by the bandwidth-delay product (BDP), where $\mathrm{BDP} = \mathrm{BW} \times \mathrm{RTT}_{\min}$. When BDP is large ($\mathrm{BDP} \geq \beta \cdot L \cdot \mathrm{MSS}$), ACK frequency is bounded by RTT. On the other hand, when BDP is small ($\mathrm{BDP} < \beta \cdot L \cdot \mathrm{MSS}$), ACK frequency is reduced proportionally to data throughput. In the work by Li *et al.*,[11] we have given the TACK frequency minimization in terms of the lower bound of β



(a)

	802.11b		802.11ac	
RTT	Delayed ACK	TACK (L=2)	Delayed ACK	TACK (L=2)
10 ms	294	294	24777	400
80 ms	294	50	24777	50
200 ms	294	20	24777	20

(b)

FIGURE 3. TACK reduces ACK frequency over the IEEE 802.11b/g/n/ac wireless links whose physical capacities are 11/54/300/866.7 Mb/s, respectively. (a) Trend of ACK frequency reduction. (b) Case study.

and the upper bound of L. By default, $\beta = 4$ and $L = 2$ have been found to be robust in practice.

According to (1) and (2), we summarize three insights as follows. First, given an L, the frequency of TACK is always no more than that of delayed ACK mechanism, i.e., $f_{\text{tack}} \leq f_{\text{delayed}}$. Second, the higher bit rate of the connection, the more number of ACKs are reduced by applying TACK. Meanwhile, the larger latency between endpoints, the more number of ACKs are reduced by applying TACK.

We further give the numeral analysis of TACK frequency over the 802.11 wireless links in comparison with the delayed ACK mechanism. Figure 3(a) reveals that more number of ACKs are reduced in the case of a faster physical capacity. Specifically, as shown in Figure 3(b), TACK has the same frequency as delayed ACK mechanism over 802.11b wireless links with a small RTT (10 ms). However, for the 802.11ac links, the frequency of TACK has dropped two orders of magnitude when RTT = 10 ms and three orders of magnitude when RTT = 80 ms. The throughput improvement of TACK due to the reduced ACK frequency was also shown in the work by Li *et al.*[11]

ROADMAP OF STANDARDIZING TACK

Applying TACK does not mean simply changing the ACK frequency according to (2). What we really want is an *on-demand* ACK mechanism that minimizes the number of ACKs but are exactly what are required by transport. The TACK-based ACK mechanism not only tells when to send ACKs, but also tells what to carry in ACKs, and how to interact with other protocol components.

When to Send ACKs?

The TACK-based ACK mechanism should answer the question that when to send which type of ACKs. Specifically, the ACK type of TACK can be triggered periodically according to (2).

Apart from the ACK type of TACK, the ACK type of Instant ACK (IACK) should also be introduced to assure timely feedback upon instant events. For example, 1) when loss occurs, the receiver should send an IACK to timely pull the desired range of lost packets from the sender. This loss-event-driven IACK enables the rapid response to loss event, effectively avoiding timeouts. 2) An IACK may be sent in the case that the received buffer nearly runs out, which assures timely send window update.

IACK and TACK are complementary. IACK assures timely and deterministic signaling while TACK acts as the last resort mechanism in the case of ACK loss.

What to Carry in ACKs?

In some cases (e.g., cellular networks), enlarging the size of ACKs will increase the backward traffic volume, cancelling out the benefit of reducing ACK frequency. However, the overhead introduced by increasing the size of ACK rather than increasing the number of ACKs can be negligible in other cases (e.g., Wi-Fi networks).

The TACK-based ACK mechanism should answer the question that what information should be carried in which type of ACKs, and why it is necessary. For example, whether an ACK should carry the information of lost packets or received packets can be considered in order to speed up recovery when loss event occurs. When the loss rate on the ACK path has reached a critical level,[11] TACK is expected to report multiple blocks in which each block reports a contiguous range of lost or received packets. TACK should also carry the ACK delay or timestamps for accurate RTT monitoring.

In addition, ACK packet encapsulation and its extension to the legacy TCP/QUIC should be specified. In order to improve space utilization, mechanisms on information compression/decompression in ACKs can also be considered.

How to Interact with Other Protocol Components?

First, the delay from a packet loss to the packet recovery is crucial to packet reassembling and, thus, impacts transport performance. The interaction between ACK mechanism and loss recovery will focus on timely loss detection on lossy forward path and robust feedback on bidirectionally lossy path.

Second, per-packet ACK achieves ideal transport state monitoring. The initial focus will be on round-trip timing under varying ACK frequency. But a receiver-based estimation framework of transport state, which acts as input of the protocol's ACK-clocking algorithms, will be included in order to reduce the estimation biases introduced by changing ACK frequency.

Third, during the startup phase, per-packet ACK is recommended to assure robust bandwidth estimation. After that, the TACK-based ACK mechanism can be applied. The congestion controller's initial focus will be on avoiding burstiness caused by lowering ACK frequency. In addition, the fairness between the flows using the TACK-based ACK mechanism and the traditional flows should also be considered.

Finally, send window update for flow control requires ACKs to update the receive window. Lowering ACK frequency probably delays acknowledging packet receipts and reporting receive window, resulting in feedback lags and wasting opportunity of sending data. The interaction with send rate control will probe into the ACK mechanism to remove these side effects.

POSSIBLE PROTOCOL MODIFICATIONS

According to the roadmap of standardizing TACK, we now discuss what are the possible protocol modifications in the context of the TACK-based ACK mechanism. As a representative of the state-of-the-art user-space deployment, the QUIC-TACK implementation is given as an example in this section.

First of all, the transport parameters should be extended for ACK mechanism negotiation. Second, multiple types of ACK frames (e.g., TACK and IACK) should be defined to support efficient loss recovery and accurate RTT monitoring. Also, a new sender-controlled frame should be defined to sync the updated TACK frequency to the receiver. Third, the algorithm of minimum RTT monitoring should be implemented in a receiver-based way. Finally, the division of labor between sender and receiver should be revisited during congestion controller modifications.

Extensions on Transport Parameters and Frames

A QUIC packet contains one or more frames. In QUIC-TACK, the TACK frames and the IACK frames may be sent over an established connection once permission has been given by QUIC's transport parameter extension of ack-frequency-support.

Ack-Frequency-Support

QUIC endpoints make declarations of their transport parameters, such as the maximum packet size, the maximum ACK delay, disable migration, etc., to negotiate features during connection establishment. In QUIC-TACK, an endpoint uses a transport parameter to signal its willingness to apply the TACK-based ACK mechanism during handshakes. If present, the transport parameter that sets ack-frequency-support is equivalent to that the TACK-based ACK mechanism is supported on the endpoint. If the transport parameter is absent, QUIC uses the default ACK mechanism.[6]

TACK Frames

The TACK frames follow the format of QUIC's legacy ACK frames (e.g., types 0x02 and 0x03).[6] That is, the ACK frames in QUIC turn to the TACK frames in QUIC-TACK, except that 1) the TACK frames are sent at the frequency of f_{tack} according to (2), and 2) the field of ACK Delay is redefined as the delay incurred between when the packet who achieves the minimum OWD is received and when the TACK is sent. The other fields carried in the TACK frame have the same meanings as defined in legacy QUIC. TACK may also define more fields according to the requirement of transport control.

A TACK frame contains one or more blocks to identify acknowledged packets, in which each block is either an ACK Range or a Gap as specified in the work by Iyengar and Thomson.[6] Each ACK Range acknowledges a contiguous range of packets, and each Gap indicates a range of packets that are not being acknowledged. QUIC-TACK reuses the loss recovery algorithm of QUIC as specified in the work by Iyengar and Thomson,[6] which is capable to achieve feedback robustness.

IACK Frames

The IACK frames (types 0x20-0x27) are newly defined in QUIC-TACK. For example, the loss-event-driven IACK (type 0x20) contains two fields of the largest packet number and the second largest packet number to indicate the most recent range of lost packets. An IACK (type 0x21) may also contain a field of the updated receive window. QUIC-TACK can reuse the existing frames (e.g., the MAX_STREAM_DATA or MAX_DATA frames) defined in legacy QUIC with a specific decision logic. Note that the type value of IACKs might change as QUIC packet structure is evolving at the IETF.[6]

ACK-FREQUENCY Frames

An ACK-FREQUENCY frame is to indicate the updated TACK frequency calculated by the sender. ACK-FREQUENCY frames are ack-eliciting. However, their loss does not require retransmission. It is recommended that the ACK-FREQUENCY frame is periodically sent during a connection (e.g., every $5-10$ RTTs).

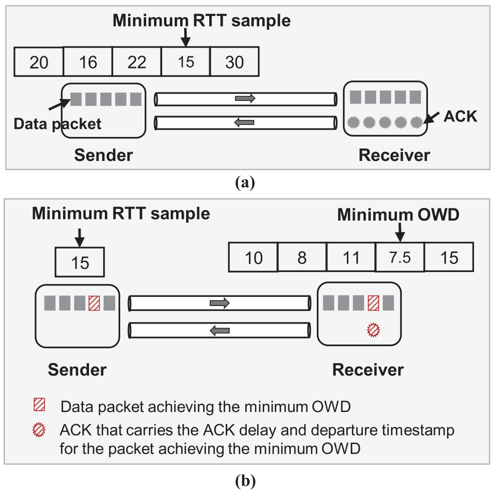

Minimum RTT sample
20
16
22
15
30
Data packet
ACK
Sender
Receiver
(a)
Minimum RTT sample
15
Minimum OWD
10
8
11
7.5
15
Sender
Receiver
Data packet achieving the minimum OWD
ACK that carries the ACK delay and departure timestamp for the packet achieving the minimum OWD
(b)

FIGURE 4. Two ways for minimum RTT monitoring. (a) Minimum RTT monitoring with per-packet ACK. (b) OWD-based minimum RTT monitoring when sending fewer ACKs.

According to (2), BW should be estimated in real time for TACK frequency update. Here BW is specified as the maximum delivery rate. The average delivery rate ($\mathrm{delivery\_rate}$) per TACK interval can be computed as the ratio of data delivered to time elapsed. At time *t*, the maximum delivery rate is a windowed max-filtered value of the delivery rates, i.e., $\mathrm{BW} = \max(\mathrm{delivery\_rate}_i)$, $\forall i \in [t - \theta_{\mathrm{filter}}, t]$, where θ_{filter} is recommended as 5–10 RTTs. The minimum RTT monitoring will be discussed next.

Redesign of Minimum RTT Monitoring

Ideally, when sending an ACK for every data packet, the minimum RTT sample can be computed by monitoring the per-packet RTT samples [see Figure 4 (a)]. However, the sender can hardly generate per-packet RTT samples in the case of sending fewer ACKs, which is the root cause of the minimum RTT monitoring biases. As shown in Figure 4(b), since the receiver is capable to monitor per-packet state, the OWD of each data packet can be easily computed according to the departure timestamp and the arrival timestamp of each data packet.

In this case, QUIC-TACK should adopt the OWD-based minimum RTT monitoring. The rationale is that the variation of OWD reflects the variation of RTT. It requires the sender to carry the departure timestamp in each ack-eliciting packet. Meanwhile, at the receiver, the per-packet OWD samples should be computed upon packet arrivals. The receiver then generates and sends a TACK frame to the sender, in which the TACK delay and departure timestamp for the packet that achieves the minimum OWD is reported. Based on the information reported by the incoming ACK frames and the TACK arrival timestamps, the sender can generate RTT samples and, then, compute the minimum RTT accordingly.

Modifications to Congestion Control

Most of the popularly used congestion controllers can work with the TACK design with corresponding implementation changes. Since lowering the ACK frequency might result in larger burstiness, in order to control the amount of sent data, QUIC-TACK should integrate with *pacing* instead of the burst send pattern. The rationale is that pacing smooth traffic behaviors by evenly spacing packets at a specific pacing rate according to the congestion controller. For example, the pacing rate may be obtained by distributing congestion window over RTT when applying a window-based controller (e.g., CUBIC[4]), and the pacing rate may also be computed using bandwidth estimate of a rate-based controller (e.g., BBR[1]).

For fairness consideration, in the work by Li *et al.*[11] we have found that the co-designed BBR in the context of the TACK-based ACK mechanism has the similar TCP friendliness to the traditional BBR in the context of the standard delayed ACK mechanism. In other words, as an ACK mechanism, our current experience shows that TACK does not impact much on the fairness of congestion controllers. However, we encourage more substantial investigations to answer the question of how does an ACK mechanism interact with various congestion controllers.

ADVANTAGES AND DISADVANTAGES

The TACK-based ACK mechanism is a good replacement of the legacy way of ACKs when the ACK overhead is nonnegligible. In the work by Li *et al.*[11] we further found the TACK-based protocol performs equally well as high-speed transport protocols in general networks (e.g., wide-area network), this is attributed to the advancements of the TACK-based protocol design in loss recovery, RTT monitoring, and send rate control.

However, the update of a novel ACK mechanism requires major intrusive modifications to the protocol. Specifically, deploying TACK upon TCP (i.e., TCP-TACK[11]) requires more efforts than deploying TACK upon QUIC (i.e., QUIC-TACK). For example, TCP-TACK needs to carry the packet number option[11] to enable the receiver-based loss detection. TCP-TACK also requires extension on the TCP data field, carrying more information (ACK Ranges or Gaps) in ACKs to achieve feedback robustness on a lossy ACK path. Different from TCP, QUIC naturally carries the packet number in data packets and allows carrying more information in ACKs, which can be reused in the QUIC-TACK implementation. Hence, a tradeoff between transport performance gain and protocol modification overhead could be considered when deploying the TACK-based ACK mechanism.

CONCLUSION

This article revisits the ACK mechanism for transport control. ACK on demand requires specifying the trigger conditions of ACKs and the information carried in ACKs according to network environment and applications. It also requires rethinking the division of labor between sender and receiver, allowing the overhead of ACKs and the efficiency of ACK-clocking algorithms to meet the transport requirements.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation for Distinguished Young Scholars of China under Grant 61825204, and in part by the Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011.

REFERENCES

1. N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2016.
2. Y. Cui, T. Li, C. Liu, X. Wang, and M. Khlewind, "Innovating transport with QUIC: Design approaches and research challenges," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 72–76, Mar./Apr. 2017.
3. G. Fairhurst, A. Custura, and T. Jones, "Changing the default QUIC ACK policy," *IETF*, 2020. Accessed: Sep. 14, 2020. [Online]. Available: https://www.ietf.org/archive/id/draft-fairhurst-quic-ack-scaling-03.txt
4. S. Ha, I. Rhee, and L. Xu, "Cubic: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.
5. J. Iyengar and I. Swett, "Sender control of acknowledgement delays in QUIC," *IETF*, 2020. Accessed: Nov. 2, 2020. [Online]. Available: https://tools.ietf.org/id/draft-iyengar-quic-delayed-ack-02.txt
6. J. Iyengar and M. Thomson, "QUIC: A UDP-based multiplexed and secure transport," *IETF*, 2020. Accessed: Sep. 25, 2020. [Online]. Available: https://tools.ietf.org/id/draft-ietf-quic-transport-31.txt
7. N. Kuhn, G. Fairhurst, J. Border, and E. Stephan, "QUIC for SATCOM," *IETF*, 2020. Accessed: Oct. 25, 2020. [Online]. Available: https://tools.ietf.org/id/draft-kuhn-quic-4-sat-06.txt
8. S. Landström and L. Larzon, "Reducing the TCP acknowledgment frequency," *ACM SIGCOMM CCR*, vol. 37, no. 3, pp. 5–16, 2007.
9. T. Li, K. Zheng, R. J., and J. Kang, "Advancing ACK handling for wireless transports,"*IETF*, 2020. Accessed: Mar. 6, 2020. [Online]. Available: https://tools.ietf.org/id/draft-li-tcpm-advancing-ack-for-wireless-00.txt
10. T. Li, K. Zheng, R. Jadhav, and J. Kang, "Optimizing ACK mechanism for QUIC," *IETF*, 2020. Accessed: Nov. 6, 2020. [Online]. Available: https://www.ietf.org/archive/id/draft-li-quic-optimizing-ack-in-wlan-01.txt
11. T. Li *et al.*, "Tack: Improving wireless transport performance by taming acknowledgments," in *Proc. ACM SIGCOMM*, 2020, pp. 15–30.
12. X. Zuo, Y. Cui, M. Wang, T. Xiao, and X. Wang, "Low-latency networking: Architecture, techniques, and opportunities," *IEEE Internet Comput.*, vol. 22, no. 5, pp. 56–63, Sep./Oct. 2018.

TONG LI is currently a Senior Researcher with the Computer Network and Protocol Lab, Huawei, Shenzhen, China. His research interest includes network protocols, security, and measurements. He received the B.S. degree from Wuhan University, Wuhan, China, in 2012, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2017. He is the corresponding author of this article. Contact him at li.tong@huawei.com.

KAI ZHENG is currently the Chief Architect and the Director of the Computer Network and Protocol Lab, Huawei, Shenzhen, China. His research interests include datacenter networking, software-defined (transport) protocols, wide-area network accelerations, Internet-of-Things protocols, and peer-to-peer communications. He received the Ph.D. degree in computer science from Tsinghua University, Beijing, China, and his Ph.D. dissertation won the first "Outstanding PhD Thesis Award" of the Chinese Computer Federation. Contact him at kai.zheng@huawei.com.

KE XU is currently a Full Professor with the Department of Computer Science & Technology of Tsinghua University, Beijing, China. He has authored/coauthored more than 200 technical papers and holds 11 US patents in the research areas of next-generation Internet, blockchain systems, Internet of Things, and network security. He received the Ph.D. degree from the Department of Computer Science & Technology, Tsinghua University. He is a member of ACM and a Senior Member of IEEE. He has guest-edited several special issues in IEEE and Springer journals. He is an Editor for the IEEE Internet of Things Journal. He is a Steering Committee Chair of IEEE/ACM International Symposium on Quality of Service. Contact him at xuke@tsinghua.edu.cn.