Revisiting Acknowledgment Mechanism for Transport Control: Modeling, Analysis, and Implementation

Tong Li, *Member, IEEE*, Kai Zheng, *Senior Member, IEEE*, Ke Xu, *Senior Member, IEEE*, *Member, ACM*, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan

***Abstract*— The shared nature of the wireless medium induces contention between data transport and backward signaling, such as acknowledgment. The current way of TCP acknowledgment induces control overhead which is counter-productive for TCP performance especially in wireless local area network (WLAN) scenarios. In this paper, we present a new acknowledgment called TACK ("Tame ACK"), as well as its TCP implementation TCP-TACK. TACK seeks to minimize ACK frequency, which is exactly what is required by transport. TCP-TACK works on top of commodity WLAN, delivering high wireless transport goodput with minimal control overhead in the form of ACKs, without any hardware modification. Evaluation results show that TCP-TACK achieves significant advantages over legacy TCP in WLAN scenarios due to less contention between data packets and ACKs. Specifically, TCP-TACK reduces over 90% of ACKs and also obtains an improvement of up to 28% on goodput. A TACK-based protocol is a good replacement of the legacy TCP to compensate for scenarios where the acknowledgment overhead is non-negligible.**

***Index Terms*— Wireless local area network, ACK frequency, Tame ACK, instant ACK.**

I. Introduction

WIRELESS local area networks (WLANs) are ubiquitous and readily getting employed in scenarios such as ultra-high-definition (UHD) streaming, VR/AR interactive gaming, and UHD IP video. The implications of video growth raise significant bandwidth demands with the video application requirements. Particularly, the peak bandwidth requirement might reach 206.9 Mbps for a 8K video [1]. However, the average WLAN connection speed worldwide (*e.g.*, 30.3 Mbps in 2018 and predicted to be 92 Mbps by 2023 [2]) is far from satisfactory for these UHD-video-based applications (see § III-A).

It is well-studied that medium acquisition overhead in WLAN based on the IEEE 802.11 medium access control (MAC) protocol [3] can severely hamper TCP throughput, and TCP's many small ACKs are one reason [4], [5]. Basically, TCP sends an ACK for every one or two packets [6], [7]. ACKs share the same medium route with data packets, causing similar medium access overhead despite the much smaller size of the ACKs [8]–[12]. Contentions and collisions, as well as the wasted wireless resources by ACKs, lead to significant throughput decline on the data path (see § III-B).

The WLAN bandwidth can be expanded by hardware modifications, such as 802.11ac and 802.11ax, in which channel binding is extended, or more spatial streams and high-density modulation are used. However, a faster physical (PHY) rate makes the MAC overhead problem even worse. This is because delay associated with medium acquisition wastes time and a higher PHY rate also proportionally increases ACK frequency for legacy TCP. There also exist some prior works that increase medium efficiency by modifying hardwares. For example, Bhartia *et al.* proposed FastACK [13] to enable the access point (AP) proactively generates fake TCP ACKs on behalf of the TCP receiver, eliminating the delay variation induced by medium contention. In contrast, this paper seeks to solve the issues without any hardware modification. We believe that rethinking the way of TCP acknowledgment that reduces medium acquisition overhead on the transport layer, so as to improve transport performance in WLAN, would be a relevant contribution.

The ACK frequency can be decreased by sending an ACK for every L ($L \geq 2$) incoming packets [9], [10], [12], [14] (*i.e.*, *byte-counting ACK*) or by sending an ACK for every large time interval (*i.e.*, *periodic ACK*). However, simply reducing ACK frequency not only impacts the packet clocking algorithms (*e.g.*, send pattern, send window update and loss detection) and round-trip timing [15], but also impairs the feedback robustness (*e.g.*, more sensitive to ACK loss). The challenge here is that legacy TCP couples the high ACK frequency with transport controls such as robust loss recovery, accurate round-trip timing, and effective send rate control (see § IV-B).

This paper presents TACK ("Tame ACK"), a type of ACK that minimizes ACK frequency by balancing byte-counting ACK and periodic ACK. To decouple the high ACK frequency from transport requirement, we propose the TACK-based

Manuscript received December 6, 2020; revised April 29, 2021; accepted July 7, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Schapira. This work was supported in part by Feng Gao. The work of Ke Xu was supported by the China National Funds for Distinguished Young Scientists under Grant 61825204, in part by the NSFC Project under Grant 61932016, and in part by the Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011. The work of Keith Winstein was supported in part by the NSF under Grant CNS-1909212 and Grant CNS-1763256 and in part by the Sloan Research Fellowship. *(Corresponding author: Ke Xu.)*

Tong Li, Kai Zheng, Rahul Arvind Jadhav, Tao Xiong, and Kun Tan are with Huawei Technologies, Shengzhen 518129, China (e-mail: li.tong@huawei.com).

Ke Xu is with the Department of Computer Science, Tsinghua University, Beijing 100084, China (e-mail: xuke@tsinghua.edu.cn).

Keith Winstein is with the Department of Computer Science, Stanford University, Stanford, CA 94305 USA.

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2021.3101011, provided by the authors.

Digital Object Identifier 10.1109/TNET.2021.3101011

acknowledgment mechanism, in which we use TACKs to sync the statistics (such as receipts, losses, available bandwidth, delays, *etc.*) between endpoints, and we also introduce Instant ACK (IACK), driven by instant events (*e.g.*, loss, state update, *etc.*), to assure timely signaling. For example, on detecting loss, an IACK will be sent to proactively pull missing packets at the receiver's buffer. TACKs and IACKs are complementary, as IACKs assure rapid feedback while TACKs assure feedback robustness (see § IV-C).

We further design TCP-TACK, the TACK-based TCP works on top of WLAN. TCP-TACK revisits the current division of labor between senders and receivers. It compensates for sending fewer ACKs by integrating the receiver-based loss detection, round-trip timing and send rate control. These components are not exactly new but are co-designed expressly to be part of the TACK-based protocol design. This cooperation between TACK and receiver-based paradigm not only minimizes the ACK frequency required, but also assures effective transport control under network dynamics (see § V and § VI).

Experiments demonstrate TACK's significant advantages over legacy way of acknowledgments in WLAN scenarios. Goodput improvement is attributed to the reduction of contention between data packets and ACKs. Furthermore, reducing ACK frequency without impacting transport performance validates the idea of decoupling high ACK frequency from transport requirement (The table in Figure 1 gives a preview of the results).

This paper is an extension of our previous conference paper [16]. In this paper, we give more details on how to implement TACK upon TCP. We also try to answer the question of how to modify the existing congestion controller (e.g., BBR [17]) when applying TACK. In addition, we add the discussion on the work-in-progress IETF drafts related to the modification of ACK mechanism. We also report our progress on the roadmap of standardizing TACK in the IETF working groups. Finally, we go deeper into some discussions such as deployment issues and experience.

	802.11b	802.11g	802.11n	802.11ac
Number of ACKs reduced	90.5%	95.4%	99.4%	99.8%
Goodput improved	20.0%	26.3%	27.7%	28.1%

Fig. 1. Percentage of goodput improvement of TCP-TACK over TCP-BBR in WLAN. Full results are in § VII-C.

II. Related Work

Reducing ACK frequency. In order to improve transport performance over IEEE 802.11 wireless links, Salameh *et al.* [5] proposed HACK by changing Wi-Fi MAC to carry TCP ACKs inside link-layer ACKs, this eliminates TCP ACK medium acquisitions and thus improves TCP goodput. We clarify the differences between TACK and HACK in three aspects. (1) TACK reduces the ACKs end-to-end while HACK only reduces the ACKs over wireless links. TACK is more general in this aspect and can be used to solve problems in asymmetric networks where the ACK path is congested [18]–[22]. (2) HACK requires network interface card (NIC) changes but no TCP changes while TACK requires TCP changes but no NIC changes. (3) Since the trigger time of the link-layer ACK and the transport-layer ACK is usually asynchronous, HACK is likely to result in ACK delays. However, HACK does not solve the transport challenges such as enlarged delay in loss recovery, biased round-trip timing, burst send pattern, and delayed send window update.

Apart from the link-layer solutions, the study of delaying more than two ACKs was first carried out by Altman and Jiménez [12], followed by a line of ACK thinning technologies [8], [9], [14], [23]–[27] on the transport layer. Among them, some studies reduce ACK frequency by dropping selected ACKs on an intermediate node (*e.g.*, a wireless AP or gateway). Due to information asymmetry, this intermediate management unavoidably makes endpoints take untimely or wrong actions. Under these circumstances, some studies adopt the end-to-end solutions, which fall into two categories: (1) Byte-counting ACK that sends an ACK for every L ($L \geq 2$) incoming full-sized packets. (2) Periodic ACK that sends an ACK for each time interval (or send window). However, both of them are far from satisfactory in the network with time-varying data rate, we will discuss this in § IV-A. This paper proposes TACK that combines these two approaches, achieving a controlled ACK frequency under different network scenarios.

Compensating for sending fewer ACKs. Compared with the studies that explore how to reduce the ACK frequency, much fewer studies explore how to compensate for sending fewer ACKs. To overcome the hurdles created by excessive ACK decrease, Allman [28] proposed 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. Landström *et al.* [10] integrated a modified fast recovery scheme and a form of the ABC algorithm to improve the TCP bandwidth utilization when ACK frequency is reduced to two or four per send window. The limitation of these algorithms, however, is that they only solve part of the problems. For example, Allman's solution did not consider the feedback robustness under excessive ACK losses. Landström's solution resulted in large router buffer occupation without smoothing the traffic bursts. Both solutions did not address the interference on the round-trip timing caused by the delayed ACKs. This paper aims to provide a complete framework that defines more types of ACKs and carries more information in ACKs, to minimize the ACK frequency required but still achieve effective feedback. In the context of TACK, this paper co-designs the receiver-based transport control to address the challenges caused by sending fewer ACKs. The receiver-based paradigm is also validated by the recent work such as pHost [29], RCC [30], ExpressPass [31], NDP [32] and Homa [33] in datacenter environments.

TACK vs delayed ACK. Transport protocols, such as TCP and QUIC [34], also alternatively adopt delayed ACK [6], [7], [35]. Delayed ACK falls into the category of byte-counting ACK except that an extra timer prevents ACK from being excessively delayed. For full-sized data packets, it turns to byte-counting ACK when bw is large and falls back to periodic ACK when bw is small (see Equation (3)). TACK differs from delayed ACK by mandatorily sending ACKs periodically when bw is large (see Equation (4)). In particular, TACK applies periodic ACK when bandwidth-delay product (bdp) is large and falls back to byte-counting ACK when bdp is small.

Both TACK and delayed ACK reduce sending ACKs. Some of the proposed ideas for compensating sending fewer ACKs in this paper, such as the advancements in round-trip timing (see § V-B), are also applicable to the delayed ACK mechanism

Video Application	SD Video	HD Video	UHD Streaming	VR	UHD IP Video	8K Wall TV	HD VR	UHD VR
Average Bit Rate (Mbps)	2	8	16	17	51	100	167	500

Fig. 2. Average bit rate of applications [2].

in the case, called "stretch ACK violation" [36], where ACKs are excessively delayed.

IETF works in progress. Some work-in-progress drafts have paid great attention to modifying the delayed ACK mechanism in the IETF QUIC working groups. For example, Fairhurst *et al.* [37] recommended reducing the ACK frequency of QUIC by sending an ACK for at least every 10 received packets and Kuhn *et al.* [38] recommended an ACK frequency of four ACKs every round-trip time (RTT), aiming to reduce link transmission costs for asymmetric paths. Instead of using an empirical value of ACK frequency, Iyengar *et al.* [39] recommended an extension of sender controlled ACK-FREQUENCY frame in QUIC to make it tunable for the frequency of the delayed ACK mechanism. In contrast to the delayed ACK mechanism that adopts the maximum value between byte-counting and a timer, TACK adopts the minimum one [40].

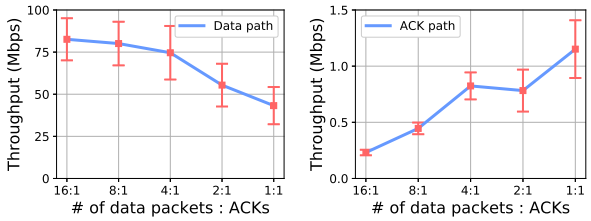
In [41] we have given a detailed discussion on the roadmap of standardizing TACK that seeks to minimize ACK frequency with corresponding improvements in transport control to compensate for sending much fewer ACKs. First, the standardization of TACK will be mainly discussed in the IETF QUIC working group since QUIC is much easier to extend. Second, instead of sending an ACK for every 10 received packets, we also recommend that QUIC adopts an adaptive ACK frequency as specified in Equation (4). Third, we recommend QUIC adopts the advanced way of calculating the minimum RTT using the relative one-way delay (OWD), reducing the information sent to the network without affecting the performance. Finally, TACK implementation in QUIC can reuse the current extensions in the QUIC working group such as Iyengar's sender controlled ACK-FREQUENCY frame to update ACK frequency.

III. Motivation

A. WLAN Demands High Throughput

It is predicted that, by 2022, the video-based applications will make up 82% of all IP traffic [42]. It is also reported that the video effect on the traffic is mainly because of the introduction of UHD video streaming [2], [43], [44]. As illustrated in the table of Figure 2, the average bit rate for UHD video at about 16 Mbps is more than 2x the high-definition (HD) video bit rate and 8x more than standard-definition (SD) video bit rate. By 2022, nearly 62% of the installed flat-panel TV sets will be UHD, up from 23% in 2017. And UHD video streaming will account for 22% of global IP video traffic. Moreover, VR/AR gaming has become increasingly popular, and the traffic will increase 12-fold, about 65% compound average growth rate per year.

It is reported that the average WLAN connection speed in 2018 was 30.3 Mbps and will be more than triple (92 Mbps) by 2023 [2]. Which, however, is still far from satisfactory for UHD-video-based applications. This is because UHD video usually requires a peak bandwidth that is multiple times of its average bit rate (*e.g.*, a video with 100 Mbps average bit rate may require over 200 Mbps peak bit rate [1]). Wireless projection is a representative UHD-video-based application.



(a) Data throughput (b) ACK throughput

Fig. 3. Examples for contention between data packets and ACKs over 802.11n wireless links.

A smartphone connects a TV using Wi-Fi Direct and streams videos on top of Miracast [45]. Our deployment experiences (see the table in Figure 16) show that UDP-based solution achieves high throughput but suffers from $5 \sim 6$ times of *macroblocking artifacts* due to unreliable transport, and legacy TCP-based solutions assure zero macroblocking but result in an over 30% of video *rebuffering ratio* [46] due to bandwidth under-utilization. Reliable and high-throughput transport over WLAN turns out to be a challenging requirement.

B. WLAN Can Be Improved on the Transport Layer

Most modern WLANs are based on the IEEE 802.11 standards. It has been well studied that the key challenge of TCP is its poor bandwidth utilization and performance when interacting with the IEEE 802.11 wireless MAC protocol [4], [5]. This can be attributed to the extensive number of medium access carried out by TCP. Basically, TCP sends an ACK every one or two packets [6], [7], which is frequent. Although the length of an ACK is usually smaller than the data packet (*e.g.*, 64 bytes for an ACK vs. 1518 bytes for a data packet), ACKs cause similar medium access overhead on the MAC layer. By sharing the same medium path for ACKs and data packets, frequent ACKs create competitions and collisions [4], [5], wasting wireless resources. As a result, the wastage leads to data rate decline on the data path. Note that although improvements in 802.11 standards (*e.g.*, 802.11ac and 802.11ax) result in data rate increase, they also cause proportionally increased number of ACKs, which makes the MAC overhead problem even worse.

To explain the problem of collision more clearly, we generated emulated traffic over the 802.11n wireless links with a PHY rate of 300 Mbps (see the table in Figure 11). It can be demonstrated that TCP's packet clocking algorithms are highly dependent on the ACK arrival pattern, and sending fewer ACKs has a negative effect on TCP throughput (see Figure 14(b)). We did not want our results to be biased because of such dependency, and hence we chose to develop our own UDP-based tool [47] that runs on two wireless laptops connected to a commercial wireless router (TL-WDR7500) with negligible external interferences. The sender keeps sending 1518-byte packets at a fixed sending rate (100 Mbps), and the receiver counts the received bytes, and then sends one 64-byte packet that act as an ACK. L emulates the byte-counting parameter that limits the amount of data to be counted before sending an ACK, *e.g.*, $L = 1$ denotes acknowledging every packet (1:1) and $L = 2$ denotes acknowledging every second packet (2:1), which are being used today and supported by IETF standards [6], [7].

As shown in Figure 3, although the throughput on the acknowledgment path is quite low (below 1.5 Mbps), the throughput on the data path decreases significantly with

Notation	Meaning
f	ACK frequency with the unit of Hz
bw	Data throughput of a connection
MSS	Maximum segment size
L	The number of full-sized data packets counted before sending an ACK
β	The number of ACKs per RTT
α	The time interval between two ACKs
RTT_{min}	The minimum RTT observed over a long period of time
bdp	The bandwidth and delay product
ρ	The packet loss rate on the data path
ρ'	The packet loss rate on the ACK path

Fig. 4. A table of notation summary.

the increase of the ACK frequency. This demonstrates that ACKs cause significant medium access overhead, degrading data transmission performance dramatically if frequent ACKs are sent. It is also observed that the ACK throughput fails to double when we raise the number of ACKs by changing the proportion between data packets and ACKs from 4:1 to 2:1. We believe that it is the result of the fierce collisions between data packets and ACKs, based on the observation of a higher bidirectional loss rate when $L \leq 2$. We also tested 802.11b/g/ac links, the insights of which remain similar.

Based on these observations, the legacy WLAN transport can be improved on the transport layer by reducing the ACK frequency required.

IV. Design Rationale

The table in Figure 4 summarizes the used notations and their meanings in this section.

A. ACK Frequency Modeling

ACK frequency can be denoted by f with the unit of Hz, *i.e.*, the number of ACKs per second. It can be reduced in two fundamental ways: byte-counting ACK and periodic ACK.

Byte-counting ACK. There exist a number of studies that reduce ACK frequency by sending an ACK for every L ($L \geq 2$) incoming full-sized packets (packet size equals to the maximum segment size (MSS)) [9], [10], [12], [14]. The frequency of byte-counting ACK is proportional to data throughput (bw):

$$f_b = \frac{bw}{L \cdot MSS} \tag{1}$$

where L indicates the number of full-sized data packets counted before sending an ACK. In general, f_b can be reduced by setting a large value of L. However, for a given L, f_b increases with bw. This means *when bw is extremely high*, ACK frequency might still be comparatively large. In other words, the frequency of byte-counting ACK is unbounded under bandwidth change.

Periodic ACK. Byte-counting ACK's unbounded frequency can be attributed to the coupling between ACK sending and packet arrivals. We therefore propose periodic ACK that decouples ACK frequency from packet arrivals, achieving a bounded ACK frequency when bw is high. The frequency of periodic ACK can be computed as

$$f_p = \frac{1}{\alpha} \tag{2}$$

where α is the time interval between two ACKs. However, *when bw is extremely low*, f_p is always as high as that in the case of high throughput. In other words, the frequency of periodic ACK is unadaptable to bandwidth change.

Delayed ACK. By default, the current transport protocols such as TCP and QUIC specify a simple delayed ACK mechanism [6], [7], [35] where 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:

$$f_{delayed} = \max\{\frac{bw}{L \cdot MSS}, \frac{1}{\alpha}\} \tag{3}$$

In practice, the delayed ACK mechanism can only reduce limited number of ACKs. As described in RFC 1122 [6] and updated in RFC 5681 [7], 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 Equation (3), when bw meets $\frac{bw}{L \cdot MSS} > \frac{1}{\alpha}$, then $f_{delayed} = \frac{bw}{L \cdot MSS}$, the higher data throughput, the higher ACK frequency, which might be unnecessary. On the other hand, when bw meets $\frac{bw}{L \cdot MSS} < \frac{1}{\alpha}$, then $f_{delayed} = \frac{1}{\alpha}$, ACK frequency cannot decrease proportionally with the decrease of bw, 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, $f_{tack} = \min\{\frac{bw}{L \cdot MSS}, \frac{1}{\alpha}\}$. In practice, α can be set to a fraction of RTT (RFC 4341 [9]), *i.e.*, $\alpha = \frac{RTT_{min}}{\beta}$. RTT_{min} is the minimum RTT observed over a long period of time, and β indicates the number of ACKs per RTT_{min}. As a consequence, the frequency of TACK is in reality given as follow:

$$f_{tack} = \min\{\frac{bw}{L \cdot MSS}, \frac{\beta}{RTT_{min}}\} \tag{4}$$

The frequency of TACK is decided by the bdp, where $bdp = bw \times RTT_{min}$. When bdp is large ($bdp \geq \beta \cdot L \cdot MSS$), ACK frequency is bounded by RTT. On the other hand, when bdp is small ($bdp < \beta \cdot L \cdot MSS$), ACK frequency is reduced proportionally to data throughput. β indicates the number of ACKs per RTT, and L indicates the number of full-sized data packets counted before sending an ACK. **Appendices B.1 $\sim$ B.2** have discussed the TACK frequency minimization in terms of the lower bound of β and the upper bound of L. By default, this paper sets $\beta = 4$ and $L = 2$ which we have found to be robust in practice (see **Appendix B.3**).

According to Equations (3) and (4), 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.*, $f_{tack} \leq f_{delayed}$. 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.

B. Challenges for Applying TACK

To apply TACK without decreasing transport performance, we list several major challenges that need to be overcome.

Enlarged delay in loss recovery. For ordered and bytestream transport, when a loss occurs and a packet has to be retransmitted, packets that have already arrived but that appear later in the bytestream must await delivery of the missing packet so the bytestream can be reassembled in order. Known as head-of-line blocking (HoLB [48]), this incurs high delay of packet reassembling and thus can be detrimental to the transport performance. Applying TACK will further enlarge this delay incurred by HoLB.

We define the *TACK delay* as the delay incurred between when the packet is received and when the TACK is sent. According to Equation (4), with a large RTT_{min}, TACK might be excessively delayed. When loss occurs during the TACK interval, the excessive TACK delay might disturb loss detection, resulting in costly retransmission timeouts. TACK loss further aggravates this problem. For example, $RTT_{min} = 200$ ms, $bw = 10$ Mbps, and $L = 1$, then $f_{tack} = 20$ Hz. Compared with per-packet ACK, TACK can cause the feedback delay up to 50 ms upon loss event. If the TACK is lost or the retransmission is lost again, then the delay doubles.

Biased round-trip timing. The initial RTT can be computed during handshakes (Figure 5 (a)), after that, the sender calculates an RTT sample upon receiving a TACK. For example in Figure 5 (b), a packet is sent at time t_0 and arrives at time t_2. Assume that the TACK is generated and sent at time t_3, the receiver computes the TACK delay $\Delta t = t_3 - t_2$. The sender therefore computes the RTT according to Δt, t_0 and the TACK arrival time (t_1), *i.e.*, $RTT = t_1 - t_0 - \Delta t$. By measuring Δt at the receiver, TACK assures an explicit correction for a more accurate RTT estimate.

The problem here is that multiple data packets might be received during the TACK interval, as shown in Figure 5 (c), generating only one RTT sample among multiple packets is likely to result in biases. For example, a larger minimum RTT estimate or a smaller maximum RTT estimate. In general, the higher the throughput, the larger the biases. One alternative way to reduce biases can be that, each TACK carries the per-packet Δt (specific TACK delays for each data packet) for the sender to generate more RTT samples. However, (1) the overhead is high, which is unacceptable especially under high-bandwidth transport. Also, (2) the number of data packets might be far more than the maximum number of Δt that a TACK is capable to carry.

Apart from loss recovery and round-trip timing, applying TACK also falls short of send rate control with regard to send pattern and send window update.

Burst send pattern. A burst of packets can be sent in response to a single delayed ACK. Legacy TCP usually sends micro bursts of one to three packets, which are bounded by $L \leq 2$ according to definition of TCP's delayed ACK [7]. However, the fewer ACKs sent, the larger the bursts of packets released. Since TACK might be excessively delayed, the burst send pattern is non-negligible as it may have a larger buffer requirement, higher loss rate and longer queueing delay if not carefully handled.

Delayed send window update. Send window update requires ACKs to update the largest acknowledged packet and the receive window (RWND). With a small frequency, TACK probably delays acknowledging packet receipts and reporting the RWND, resulting in feedback lags and bandwidth under-utilization. For example, $f_{tack} = 20$ Hz, then TACK is sent every 50 ms. Assume a TACK notifies RWND $= 0$ due to receive buffer runs out at $t = 0$ ms, upon receiving this TACK, the sender stops sending data. In the case that the receive buffer is released at $t = 5$ ms due to loss recovery, the sender continues to be blocked for another 45 ms until a subsequent TACK is sent at $t = 50$ ms, and thus wastes opportunity of sending data. TACK loss further aggravates this issue.

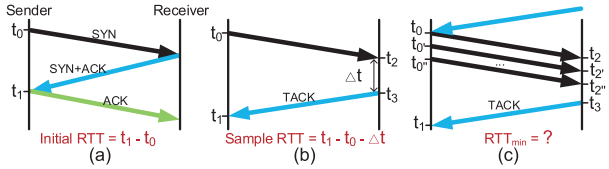


Fig. 5. TACK-based round-trip timing: a case study.

C. TACK-Based Acknowledgment Mechanism

Applying TACK significantly reduces ACK frequency. However, as discussed above, independently using TACK probably falls short of robust loss recovery, accurate round-trip timing, and effective send rate control. What we really want, for WLAN, is a full TACK-based acknowledgment mechanism that overcomes the hurdles for applying TACK, using a controlled frequency of ACKs to support efficient transport.

There are some notable features of the TACK-based acknowledgment mechanism which are important for reasoning about the differences from legacy TCP. We briefly describe these features below.

More types of ACKs. Apart from the ACK type of TACK, we also introduce the ACK type of IACK ("Instant ACK") to assure timely feedback upon instant events. For example, (1) when loss occurs, the receiver sends 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 receive buffer nearly runs out, which assures timely send window update. In addition, (3) the sender might send an IACK to sync an updated TACK frequency with the receiver for adjusting TACK interval.

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 (§ V-A). Note that both sender and receiver can send IACKs on demand.

More information carried in ACKs. First of all, reducing ACK frequency may require extending TACK to carry more information if the link has deteriorated. For example, to reduce feedback delay under excessive ACK loss, TACK is expected to report as many blocks as possible, in which each block reports a contiguous range of lost or received packets (see § V-A). It is worth mentioning that this rich information can be carried on demand. Specifically, only when the loss rate on the ACK path has reached a critical level (see Equation (5)) will carrying more information be profitable.

TACK might also be required to carry the TACK delay for accurate RTT estimation and might carry timestamps if latency such as one-way delay is computed at the sender. Furthermore, although the sender can achieve an approximate computation accuracy of some transport states, such as delivery rate, congestion window and loss rate, the receiver-based computation is more straightforward in the context of a reduced ACK frequency. Optionally, by shifting these functionalities from sender to receiver and syncing results through TACKs, the total

CPU and memory usages at both endpoints might be reduced at the cost of the larger size of TACKs.

Note that carrying more information in TACK does not introduce excessive overhead over WLAN, as it only increases the size of ACK rather than increasing the number of ACKs. We believe the improved feedback robustness will more than pay for the TACK extension overhead.

Less number of ACKs. Although adopting more types of ACKs, we still have the advantages of significantly reducing ACK frequency in most cases. This is because the event-driven IACK is rarely triggered, whose frequency is usually low and negligible. For example, with a packet loss rate (ρ), the highest frequency of the loss-event-driven IACK is $\frac{\rho \cdot bw}{MSS}$, this type of IACK only adds a small number of ACKs on the return path in practice. First, the frequency of IACK is much lower than $\frac{\rho \cdot bw}{MSS}$ because packets are usually lost consecutively. Second, ρ is usually a small percentage (*e.g.*, $< 10\%$).

V. TACK-Based Protocol Design

This section introduces the detailed design of TACK-based protocols, in which the advancements in loss recovery, round-trip timing, and send rate control are the most key reasons that the dependence on frequent ACKs has decreased.

A. Advancements in Loss Recovery

Instead of the traditional reactive approach where the sender counts duplicate ACKs or analyzes packet timestamps, a TACK-based protocol adopts a receiver-based loss detection, in which the packet number, the IACK, and the TACK play different roles.

Packet number enables receiver-based loss detection. First of all, we must overcome the so-called "retransmission ambiguity", which refers to the fact that the receiver cannot accurately identify the number of retransmission losses when employing the TCP sequence numbering scheme [35]. In this paper, we introduce a monotonically increasing number for each packet, *i.e.*, the *packet number* (PKT.SEQ). Therefore, a data packet contains both sequence number (SEQ) and packet number. SEQ is the existing data sequence number used in legacy TCP to assure bytestream can be reassembled in order. PKT.SEQ directly encodes the transmission order. In other words, a packet sent later owns a higher PKT.SEQ than the packet sent earlier. When a packet is being retransmitted, both of its payload and SEQ remain the same while its PKT.SEQ is updated. PKT.SEQ removes the ambiguity about which packet is lost when losses are detected.

To explain this clearly, we give an example where 5 packets with bytestream range $[0 \sim 5999]$ are sent (MSS=1500 bytes). Assume packet $[1500 \sim 2999]$ with PKT.SEQ $= 2$ is dropped, when subsequent packet $[3000 \sim 4499]$ with PKT.SEQ $= 3$ arrives, the receiver detects loss and informs the sender to retransmit $[1500 \sim 2999]$ with PKT.SEQ $= 4$. Assume the retransmitted packet with PKT.SEQ $= 4$ is dropped again, when subsequent packet $[4500 \sim 5999]$ with PKT.SEQ $= 5$ arrives, the receiver is still able to detect the retransmission loss. However, without packet number, receiver-based loss detection can hardly detect the exact number of lost retransmissions.

We note that, for the TACK-based protocol, the sender has to maintain a two-tuples (SEQ, PKT.SEQ) for each packet. Although retransmissions will have different PKT.SEQs, for implementation it is recommended the PKT.SEQ of a packet in the tuples be always replaced and updated by the latest PKT.SEQ of the retransmitted packet. This is reasonable since the packet with a smaller PKT.SEQ has already been retransmitted, the sender does not need to maintain extra state to check on whether this packet has been received. In this case, the extra overhead by introducing the packet number turns out to be negligible. The packet number in TACK is semantically similar to the packet number as specified in QUIC [34].

IACK speeds up loss recovery on lossy data path. The legacy TCP sends per-packet ACK when loss occurs, in contrast, our design only sends a single IACK. Loss-event-driven IACK is a supplemental method for TACK to assure rapid reaction to loss events, significantly reducing feedback delay of TACK. The IACK determines losses according to the out-of-order packets in the PKT.SEQ space.[1] Specifically, the IACK integrates two fields, the largest PKT.SEQ and the second largest PKT.SEQ of the received packets, to indicate the most recent range of lost packets, with which the sender can retransmit lost packets timely upon IACK arrivals. Considering the above example, assume packet with PKT.SEQ $= 1$ is received and PKT.SEQ $= 2$ is dropped, upon packet with PKT.SEQ $= 3$ arrives, an IACK contains PKT.SEQ $= 1$ and PKT.SEQ $= 3$ is constructed according to the out-of-order delivery, notifying the sender of the loss of PKT.SEQ $= 2$.

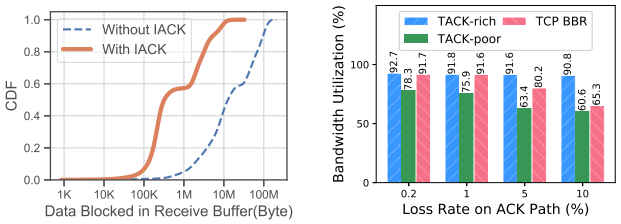
To investigate how the loss-event-driven IACK impacts loss recovery, we randomly sample the packet loss rate between 0 and 3% on data path, and the RTT between 1 and 200 ms. We report the amount of data blocked in the receiver's buffer at the time when a TACK is sent. Figure 6(a) shows the results. It is demonstrated that IACK decreases the delay incurred by HoLB, as a result, the memory pressure is significantly reduced at the receiver.

It is worth noting that the loss-event-driven IACK shares the same idea as the "negative ACK" (NACK or NAK), which has been widely used in error-control mechanisms for data transmission (*e.g.*, WebRTC [49], UDT [50], RBUDP [51], NACK Option [52], and NORM [53]). However, the proposed IACK in this paper is a novel concept of acknowledgment whose formation is triggered by an instant event. These instant events can be not only an event of packet loss, but also an event that buffer runs out, an event of RTT update request and so on.

TACK assures loss recovery robustness on bidirectionally lossy path. When losses are only on the data path, the delay incurred by HoLB can be decreased by timely sending loss-event-driven IACKs. However, on a bidirectionally lossy path, loss notification of IACK might also be lost. To bound the delay incurred by HoLB, *proactively* and *periodically*, TACK carries rich information to pull lost packets and also acknowledges packet receipts.

Specifically, the information carried in TACK contains ranges of packets which are *alternately* in the "*acked list*" and in the "*unacked list*". The "acked list" is a list of the blocks of contiguous packets that have been received and queued at the receiver, and the "unacked list" is a list of the gaps between the non-contiguous blocks of data that have been received and queued at the receiver. For example, packets 1 to 10 are sent and packets 1, 4, 5, 6, 10 are received. In this case, the "acked list" can be the blocks of {1}, {4, 6}, and {10}, and the "unacked list" can be the blocks of {2, 3}, {7,9}. Limited by MSS, a TACK might not be able to carry

[1] An IACK is sent right away by default, however, it should be slightly delayed in the case of reordering. We will discuss this in § VIII.



(a) Lossy data path (b) Bidirectionally lossy path

Fig. 6. Loss recovery in the context of TACK. (a) IACK reduces memory pressure at the receiver. (b) Carrying rich information in TACK assures high bandwidth utilization.

all blocks. In principle, TACK should preferentially carry the blocks with the largest serial number in the "acked list" (*e.g.*, {10}), or the blocks with the smallest serial number in the "unacked list" (*e.g.*, {2, 3}).

One of the high level idea of reducing the ACK frequency is using more informative ACK. ρ denotes the packet loss rate on the data path, and Q denotes the primary number of blocks in the "unacked list" that a TACK has reported. It can be derived that more information should be carried when the loss rate (ρ') on the ACK path follows:

$$\rho' > \begin{cases} \dfrac{Q \cdot MSS}{\rho \cdot bdp}, & bdp \geq \beta \cdot L \cdot MSS \\ \dfrac{Q}{\rho \cdot L}, & bdp < \beta \cdot L \cdot MSS \end{cases} \tag{5}$$

Furthermore, the additional number of blocks ($\triangle Q$) in the "unacked list" that the TACK should report is given by $\triangle Q = \frac{\rho \cdot \rho' \cdot bdp}{MSS} - Q$ when $bdp \geq \beta \cdot L \cdot MSS$, and $\triangle Q = \rho \cdot \rho' \cdot L - Q$ when $bdp < \beta \cdot L \cdot MSS$. Please refer to **Appendix A** for detailed derivation.

To investigate how the rich information in TACK improves performance, we transmit a long-lived flow on a bidirectionally lossy path with the RTT of 200 ms (refer to § VII-A for testbed setup). "TACK-poor" refers to the TACK-based TCP implementation (§ VI) that only acknowledges the largest ordered packets accumulatively and reports the smallest out-of-order packets in the receive buffer (*i.e.*, $Q = 1$), and "TACK-rich" refers to the version that reports as many losses and receipts as possible in each TACK. Both employ BBR [17] as congestion controller. We set a constant loss rate (1%) on the data path, and varying loss rates on the ACK path. Figure 6 shows the results. "TACK-poor" suffers from throughput decline in the case of ACK loss. It also demonstrates that TCP BBR's margin benefits from the SACK option [54] and RACK [55] decrease with the increase of the ACK loss rate. In contrast, "TACK-rich" repeatedly carries rich information in TACK to improve loss recovery robustness, making transport insensitive to bidirectional losses. Note that the utilization of "TACK-rich" is barely decreased with the high ACK loss (10%), this can be attributed to the more number of blocks (> 4) reported by a TACK, while TCP's SACK option only reports 3 or 4 blocks per ACK [54].

Note that in order to avoid unnecessary retransmission, TACK only reports missing packets that have been reported by loss-event-driven IACKs, while the sender only retransmits a specific packet once per RTT when the loss is repeatedly notified by both IACKs and TACKs.

B. Advancements in Round-Trip Timing

As discussed above, legacy way of round-trip timing adopts simple RTT sampling (§ IV-B), introducing either large biases or high overhead for large bdp transport. Without loss of generality, this section takes the minimum RTT estimation as an example.[2] Aiming to reduce TACK's overhead of accurate round-trip timing, we propose a receiver-based way to estimate the minimum RTT *indirectly* without maintaining too many connection states.

The rationale is that the variation of one-way delay (OWD) reflects the variation of RTT. The OWD estimation does not require clock synchronization here as we use relative values. For example in Figure 5 (c), a relative OWD sample can be computed as $OWD = t_2 - t_0$, where t_0 and t_2 are the packet departure timestamp and the packet arrival timestamp, respectively. Upon packet arrivals, the receiver is capable to generate per-packet OWD samples.

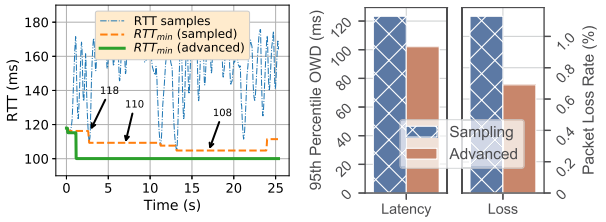
The smoothed OWD is an exponentially weighted moving average (EWMA) [56] of the per-packet OWD samples at the receiver. According to the smoothed OWD, the minimum OWD during each TACK interval can be observed. Afterwards, based on the TACK delay ($\Delta t^{\star}$) and the departure timestamp ($t_0^{\star}$) corresponding to the packet that achieves the minimum OWD, the sender calculates the RTT of this packet as a minimum RTT sample. Ultimately, the minimum RTT is computed according to these minimum RTT samples using a minimum filter [17], [57] over a long period of time τ ($\tau \leq 10$ s), where the 10-second part is to handle route changes. Note that we adopt two minimum filters at both sides because the minimum filter at the sender further implicitly reduces biases of the ACK delivery.

To investigate how round-trip timing impacts performance, we first discuss the accuracy of RTT_{min} as a microbenchmark that we seek to improve. We use the TACK-based TCP implementation (§ VI) to transmit flows between two Wi-Fi endpoints, with a network emulator forwarding. A fixed bidirectional latency (100 ms) is set between the endpoints. Figure 7(a) shows that the advanced round-trip timing tracks the real minimum RTT. However, legacy RTT sampling suffers $8\% \sim 18\%$ larger RTT_{min} estimates. We further explore performance improvement on real paths over the Internet [58]. As illustrated in Figure 7(b), applying the advanced round-trip timing has reduced 20% of the 95th percentile OWD and 54% of the packet loss. Note that this improvement is obtained without sacrificing throughput [59], [60]. We infer that an accurate minimum RTT estimate avoids pushing too much data into the pipe, and thus reduces latency and loss.

C. Advancements in Send Rate Control

Lowering the ACK frequency might result in larger burstiness. In order to control the amount of sent data, TACK-based congestion controller should integrate with *pacing* instead of the burst send pattern. The rationale is that pacing [61] smooths traffic behaviors by evenly spacing packets at a specific pacing rate (denoted by $pacing\_rate$) according to the congestion controller. For example, $pacing\_rate$ may be obtained by distributing congestion window (CWND) over RTT when applying a window-based controller (*e.g.*, CUBIC [62]), and $pacing\_rate$ may also be computed using bandwidth estimate of a rate-based controller (*e.g.*, BBR [17]).

[2] In general, the minimum RTT estimation can be easily extended to the x^{th} percentile RTT estimation, where $x \in (0, 100]$.



(a) Microbenchmark (b) Wild Internet tests

Fig. 7. Round-trip timing in the context of TACK. (a) Legacy RTT sampling suffers $8\% \sim 18\%$ larger RTT_{min} estimates. (b) Latency and loss change before [59] and after [60] applying the advanced round-trip timing.

The $pacing\_rate$ can be computed at both endpoints. Take the rate-based controller as an example, if a BBR-like bandwidth estimation [17] is adopted, the pacing rate at time t is computed at the sender side using a windowed max-filter (θ_{filter} is set at several RTTs): $pacing\_rate_t \propto max(delivery\_rate_i),\ \forall i \in [t - \theta_{filter}, t]$, where $delivery\_rate_i$ is the deliver rate computed upon each TACK arrival. Since receiver-based computation is more straightforward than sender-based one in the context of TACK, a rate-based controller may conduct bandwidth estimation in a receiver-based way instead, *i.e.*, the $delivery\_rate$ is computed at the receiver upon data packet arrivals and synced to the sender via TACK. With regard to the window-based controllers such as CUBIC, Vegas [63], and Compound TCP [64], a TACK-based congestion controller requires converting the CWND to the pacing rate [57]: $pacing\_rate_t \propto \frac{CWND}{sRTT_t}$, where $sRTT_t$ denotes the smoothed RTT at time t.

Most of the popularly used congestion controllers can work with TACK by handling minor changes. Moreover, rate-based congestion controllers (*e.g.*, BBR) usually requires less changes than window-based ones (*e.g.*, CUBIC). In this paper, we give an example for the design and implementation of a TACK-based congestion controller co-designing BBR. BBR's RTT and bandwidth estimations are all coupled with frequent ACKs. However, these can be implemented with small amount of work by moving the estimation logic from sender to receiver. This receiver-based paradigm also fits the TACK-based acknowledgment mechanism well. On the other hand, since one round of pacing rate control can be as large as multiple RTTs (*e.g.*, 8), BBR is supposed to work well with the TACK-based protocol framework where ACKs are excessively delayed. We will give the detailed receiver-based BBR implementation in § VI-D. In addition, we further conducted experiments to demonstrate that the receiver-based BBR in the context of TACK performs similar to the legacy BBR (see **Appendix C**).

In addition, lowering ACK frequency probably causes bandwidth under-utilization without timely updating the send window. To tackle this issue, an IACK updating the largest acknowledged packet and the RWND should be sent without delay when encountering an abrupt change of receive buffer. For example, when the receive buffer usage is full, an IACK may be generated to report a zero window.

VI. Protocol Implementation

TACK, or its acknowledgment mechanism, can be implemented in most of the ordered and reliable transport protocols. This paper mainly discusses TCP-TACK, a TACK-based TCP implementation that applies TACK and deploys the advancements as specified in § V-A∼V-C. A full implementation of TCP-TACK including all the above advancements requires extension on the TCP option to introduce more ACK types, and also requires extension on the TCP data field to carry more information in ACKs.

Type	Length	Name	Description
40	2 bytes	TACK-Permitted	In a SYN packet to enable TACK scheme
41	3 bytes	ACK-Type	Indicates ACK types
42	6 bytes	PKT.SEQ	In a data packet to indicate packet number

Fig. 8. TCP option extension using TLV encoding.

The legacy TCP adopts the TCP option extension to carry newly defined information such as the Timestamp option, the SACK option, the MSS option, *etc.* TCP-TACK can also be implemented by defining more option fields for more ACK types and the rich information (*e.g.*, TACK delay, "unacked list", delivery rate, *etc.*) carried by ACKs. However, due to the 40-byte limitation of the TCP option, TCP-TACK might fail to report enough number of ACK blocks (*e.g.*, more than 4) in the case of bidirectionally lossy paths.

In this case, we first define more types of ACKs by extending the TCP options in the TCP header, and then extend the TCP data field to carry more information in ACKs. This design rationale comes from three concerns: First, the TCP option extension makes full use of the residual header space. Second, the data field extension only increases the size of ACKs rather than increasing the number of ACKs. Ultimately, piggybacking acknowledgment [65] might be unnecessary because TCP-TACK's ACK frequency is usually already extremely low. This greatly motivates the data field extension in TCP-TACK.

A. TCP Option Extension

The TACK option. As illustrated in the table of Figure 8, the TACK option uses two TCP alternatives with type-length-value (TLV) encoding [54]. The first is an enabling option, `TACK-Permitted`, which may be sent in a SYN (synchronization) packet to indicate that TACK-based acknowledgment mechanism can be used once the connection is established. The second is an ACK indicator option, `ACK-Type`, whose value indicates the type of ACKs (*e.g.*, TACK $= 0 \times 01$, IACKs $= 0 \times$ 02-0xff) over an established TCP connection once permission has been given by the `TACK-permitted`.

The packet number option. A TCP-TACK packet contains both sequence number (SEQ) and packet number (PKT.SEQ). SEQ is the existing data sequence number used in legacy TCP, and TCP-TACK further extends the `PKT.SEQ` option in each data packet for receiver-based loss detection. As shown in the table of Figure 8, the option contains a 4-byte value for the packet number and it is sent over an established connection once permission has been given by the `TACK-permitted`.

B. Data Field Extension

Extension for loss recovery. Figure 9 gives an example of the format of the loss-event-driven IACK (type 0×02), which contains two fields to indicate the most recent range of lost packets in the PKT.SEQ space. The `Largest PKT.SEQ` and the `Second largest PKT.SEQ` refer to the largest PKT.SEQ and the second largest PKT.SEQ of the received packets, respectively. IACKs also accumulatively acknowledge ordered packets in the SEQ space by reusing the field of `Acknowledgment Number` in the TCP header [66].

TACK syncs rich information between endpoints. The data field of TACK can be extended on demand as long as the

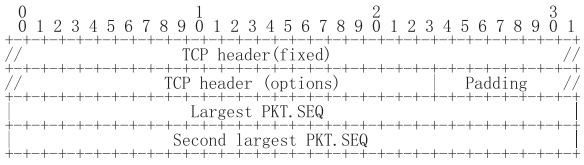


Fig. 9. Loss-event-driven IACK format in TCP-TACK.

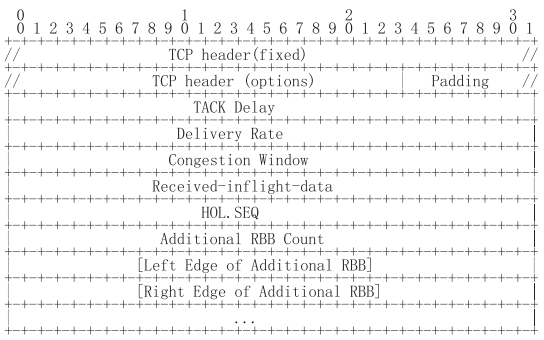


Fig. 10. TACK format in TCP-TACK.

TACK size does not exceed the MSS (*e.g.*, 1500 bytes). Figure 10 shows an example for the TACK format.

In order to bound the delay incurred by HoLB under ACK losses, we further define *receive-buffer-bubble* (RBB) in the "black list" as the gap between non-contiguous blocks of *SEQ space* occupied by data that has been received and queued at the receiver. TACK defines `HOL.SEQ` for the smallest SEQ of out-of-order packets in the receive buffer, and acknowledges the largest ordered packets accumulatively by reusing the field of `Acknowledgment Number` in the TCP header [66]. These two SEQs report the first RBB to pull packets that are the most required according to the delivery ordering.

Under excessive bidirectional losses, pulling only one RBB per TACK can be inefficient (Figure 6). Optionally, TACK further defines a series of additional `RBB` fields for redundant loss feedbacks. No more than MSS in length, one TACK can carry hundreds of RBBs. Upon a packet arrival, RBBs are updated with the complexity of $O(logN)$. A TACK should report the *first a few* RBBs. Specifically, when the receiver sends a TACK, the following two rules apply. (1) A TACK should include as many distinct RBBs as possible. (2) A TACK should be filled out by repeating the oldest RBBs (with the smallest SEQs). This assures every RBB can be reported multiple times.

Extension for round-trip timing. TACK redefines the TCP `Timestamp Option` [66], such that *TSval* indicates the timestamp when the packet who achieves the minimum OWD is sent, and *TSecr* indicates the timestamp when TACK is sent. TACK also defines a new field of `TACK Delay` to indicate the delay incurred between when the packet who achieves the minimum OWD is received and when the TACK is sent.

Extension for send rate control. For bytes-in-flight update, TACK defines `Received-inflight-data` for the amount of data that has been received but not acknowledged. TACK also defines more fields such as `Congestion Window` for the window-based congestion controllers and `Delivery Rate` for the rate-based congestion controllers. Both TACK and IACK reuse the field of `Window Size` in the TCP header [66] for the receive window advertisement.

C. Transport State Monitoring

A frequency-update-driven IACK is to indicate the updated TACK frequency calculated by the sender. It is recommended that the frequency-update-driven IACK is sent when the updated TACK frequency doubles or halves. In this paper, bw is specified as the maximum delivery rate. The average delivery rate ($delivery\_rate$) per TACK interval can be computed as the ratio of data delivered to time elapsed. Details are given in § VI-D. The minimum RTT (RTT_{min}) can be monitored according to § V-B. Both bw and RTT_{min} should be monitored in real time for TACK frequency update according to Equation (4).

Moreover, the receiver computes the loss rate (ρ) on the data path per TACK interval. ρ is the ratio of number of lost packets to number of packets that should have been received. The sender also computes the loss rate (ρ') on the ACK path when RTT_{min} is updated. ρ' is the ratio of number of lost TACKs to number of expected TACKs during a period of time. These transport state are monitored to decide if more information should be carried in ACK according to Equation (5).

D. Receiver-Based BBR Implementation

This section discusses the implementation of a receiver-based BBR as the congestion controller for TCP-TACK.

The receiver-based BBR adopts pacing, instead of the burst send pattern, for every data packet at a pacing rate that is updated upon each TACK arrival. After sending at a certain rate and waiting for one TACK interval, the sender updates its sending rate according to the maximum bandwidth estimation of a newly coming TACK.

Similar to legacy BBR [17], a flow adopting the receiver-based BBR has three states: *Startup*, *Drain*, and *Stable*. The *Startup* and *Drain* states usually last for a few TACK intervals. Note that it is suggested to use the legacy TCP's per-packet ACK to speed up the bandwidth probing in the startup phase, and meanwhile, this removes the side effect that may appear in the case of short flows. The *Stable* is the main state that covers two events: maximum bandwidth estimation (*ProbeBW*) and minimum RTT estimation (*ProbeRTT*). Both estimations are receiver-driven and relatively independent from the ACK delivery.

ProbeBW. To reduce the interference of burst, the receiver updates the delivery rate ($delivery\_rate$) per TACK but calculates as $delivery\_rate = \frac{\Delta delivered \cdot f_{tack}}{2}$ when the system is not application-limited. $\Delta delivered$ refers to the bytes received during two TACK intervals (usually approximates the OWD). At time t, the maximum available bandwidth (BW_{max}) is a windowed max-filtered value of the delivery rates carried in TACKs, *i.e.*, $BW_{max} = \max(delivery\_rate_i),\ \forall i \in [t - \theta_{filter}, t]$, where the bandwidth filter window θ_{filter} is set at several dozens according to the steady phase cycle. The pacing rate is computed as $pacing\_rate_t = pacing\_gain \cdot BW_{max}$, where $pacing\_gain$ is a control parameter that varies in different phases as discussed next.

The *ProbeBW* lasts for dozens of TACK intervals and contains three subphases: the *gain* cycle occupying four TACK intervals, the *drain* cycle occupying four TACK intervals, and the rest cycles are *cruise* cycles. The recommended duration of *ProbeBW* is 32 TACK intervals, which approximates BBR's 8 RTTs. Accordingly, the bandwidth

Link	Spatial streams	Modulation type	Coding rate	Guard interval	Channel width	PHY capacity	UDP baseline≈
802.11b	-	CCK	-	-	22MHz	11 Mbps	7 Mbps
802.11g	-	64-QAM	3/4	-	20MHz	54 Mbps	26 Mbps
802.11n	2	64-QAM	5/6	400ns	40MHz	300 Mbps	210 Mbps
802.11ac	2	256-QAM	5/6	400ns	80MHZ	866.7 Mbps	590 Mbps

Fig. 11. Parameters of 802.11-based links.

filter window θ_{filter} is set 40 TACK intervals (approximates 10 RTTs) to cover feedback delay. The pacing rate is computed as $\max(pacing\_gain \cdot BW_{max}, MSS \cdot f_{tack})$, where $pacing\_gain = 1.25$, 0.75, and 1 during the *gain*, the *drain*, and the *cruise* cycles, respectively.

ProbeRTT. *ProbeRTT* is entered when RTT_{min} has not been updated by a lower measured value for several seconds (*e.g.*, 10 s). The sender limits its pacing rate to $MSS \cdot f_{tack}$ for $\max(RTT, 200\ ms)$. This aims to drain the queue to enable the minimum RTT estimation.

Inflight cap. BBR does not use a congestion window or ACK clocking to control the amount of inflight data, instead, it uses an inflight data limit of $inflight\_cap = cwnd\_gain \cdot bdp$, where $bdp = BW_{max} \times RTT_{min}$ and $cwnd\_gain$ is a control parameter. Upon a TACK arrival, the sender updates the bytes in flight ($inflight\_size$) by subtracting the received bytes from the sent bytes. The received bytes can be updated upon TACK arrivals. For example, a TCP-TACK receiver directly carries a field of `Received-inflight-data` in a TACK (see Figure 10).

Before sending a packet, the bytes in flight must satisfy $inflight\_size < inflight\_cap$, otherwise the sender has to wait. During the *Stable* state, legacy BBR sets $cwnd\_gain = 2$, which allows BBR to continue sending smoothly at the pacing rate even when ACKs are delayed by one RTT [17]. In general, TCP-TACK delays the ACK at most for one TACK interval, and we therefore set $cwnd\_gain = 2 + \frac{1}{RTT_{min} \cdot f_{tack}}$. Note that we have also integrated the improvements specified in [67] to improve throughput over Wi-Fi network paths with aggregation.

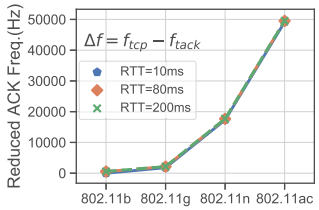
VII. Evaluation

In this section, we first give numeral analysis of TACK frequency over the 802.11 wireless links. We then investigate the ideal and the actual performance of TCP-TACK in WLAN scenarios, including the deployment experience in commercial products. We finally investigate how TCP-TACK would work over the combined links of WLAN and wide area network (WAN).

A. Experiment Setup

Experiments are conducted on various wireless links (*e.g.*, IEEE 802.11b/g/n/ac), controllable links connected with a Spirent Attero network emulator [68], and shared links on the Internet, using the link conditions for randomized experimental trials. If not otherwise specified, the PHY raw bit rates of 802.11b/g/n/ac links are 11/54/300/866.7 Mbps, respectively. Detailed parameters are listed in the table of Figure 11.

Since this paper mainly discusses acknowledgment mechanism rather than congestion control, we do not intend to investigate the differences among various congestion controllers. Instead, we focus on the comparison between different acknowledgment mechanisms in the context of the same congestion controller upon the same transport protocol. Particularly, TCP-TACK is compared with TCP BBR. TCP-TACK



(a) ACK frequency reduction

RTT_{min}	802.11b TCP (L=2)	802.11b TACK (L=2)	802.11ac TCP (L=2)	802.11ac TACK (L=2)
10 ms	294	294	24777	400
80 ms	294	50	24777	50
200 ms	294	20	24777	20

(b) ACK frequency (Hz)

Fig. 12. TACK reduces ACK frequency over the IEEE 802.11b/g/n/ac wireless links.

is implemented upon the TCP of our user-mode Stack based on the Netmap framework [69]. Unless otherwise noted, TCP-TACK refers to the TCP-TACK that implements the receiver-based BBR.

The Linux kernel follows the TCP [6], [7] guidelines of sending an ACK for every second full-sized data packet received. For experimentation we changed the Linux Kernel 5.3 TCP code [70] to allow the receiver sends an ACK for every L ($L \geq 2$) full-sized data packets, with which we can deploy prior ACK thinning mechanisms, *i.e.*, TCP variants with $L = 4, 8, 16$. We introduced a new option called, `BPF_SOCK_OPS_ACK_THRESH_INIT`, as part of the BPF socket options [71] (`BPF_PROG_TYPE_SOCK_OPS`) to allow changing the ACK frequency. This option operates in TCP control flow handling only and does not introduce any runtime overhead during data flow.

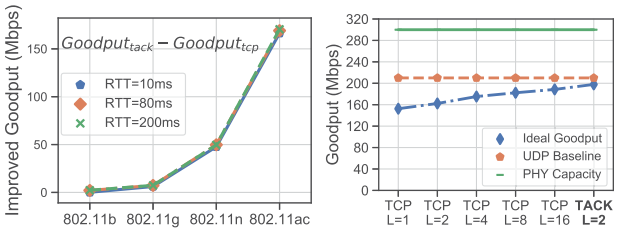
For a fair comparison, we tried our best to use default versions and parameters for all schemes. For example, TCP BBR represents TCP using BBR as congestion controller and RACK [55] as loss detection algorithm. TCP CUBIC is the default SACK-enabled implementation in the latest Linux kernels. Unless otherwise noted, TACK sets $L = 2$, TCP delayed ACK is enabled, and data packets are full-sized with $MSS = 1500$ bytes.

B. TACK Frequency in Real-World Deployments

First of all, we give numeral analysis of TACK frequency over the 802.11 wireless links in comparison with standard delayed ACKs. Figure 12(a) shows that more number of ACKs are reduced in the case of a faster PHY capacity. Specifically, as shown in Figure 12(b), TACK has the same frequency as TCP's delayed ACK (denoted by TCP (L=2)) over 802.11b wireless links with a small RTT_{min} (10 ms). However, for the 802.11ac links, the frequency of TACK has dropped two orders of magnitude when $RTT_{min} = 10$ ms and three orders of magnitude when $RTT_{min} = 80$ ms. Note that Figures 12(a) and 13(a) also reveal that goodput increase is insensitive to the latency between endpoints. This is because TACK's frequency is already quite low, reducing ACK frequency by hundreds of Hz only slightly impact goodput.

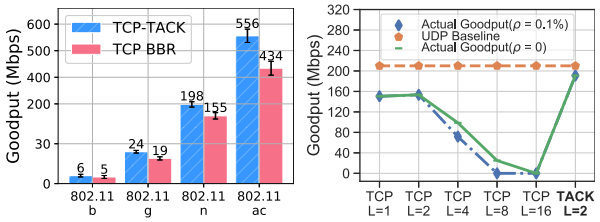
C. Performance in WLAN Scenarios

Before diving into protocol performance, we first answer the question of how close TACK can get to transport upper bound. We use the UDP-based tool [47] specified in § III-B to estimate the ideal goodput of different ACK thinning techniques. For example, "TCP (L=8)" considers the case of byte-counting ACK that sends an ACK for every 8 packets. The emulator keeps sending 1518-byte packets from the sender, the receiver counts 8 received packets, and then sends one 64-byte packet



(a) Goodput improvement (b) Ideal goodput trend

Fig. 13. (a) Links with faster PHY rate enlarge goodput improvement. Note that $Goodput_{tcp}$ here refers to the goodput of TCP without "negative effect", whose traffic is emulated by UDP. (b) TACK approaches the transport upper bound with a minimized ACK frequency (RTT=80 ms, 802.11n).



(a) TCP-TACK vs. TCP BBR (b) Actual goodput trend

Fig. 14. (a) TCP-TACK obtains 20% $\sim$ 28.1% of goodput improvement. (b) Prior ACK thinning mechanisms impact the TCP transport performance (RTT=80 ms, 802.11n).

as an ACK. "UDP Baseline" refers to the goodput of UDP without sending any ACKs, it acts as the transport upper bound as its goodput is not impacted by ACKs. "PHY Capacity" is the raw bit rate at the PHY layer. For IEEE 802.11n links, "UDP Baseline" = 210 Mbps and "PHY Capacity" = 300 Mbps according to the table in Figure 11.

It is well-known that the transport performance will be impacted when the number of ACKs is excessively reduced. Thus, sending fewer ACKs has a "negative effect" on the transport performance. However, in wireless scenarios sending fewer ACKs also has a "positive effect" on the transport performance due to the reduced contentions. To better estimate this "positive effect", we assume that there is no "negative effect" ideally. As a result, "Ideal Goodput" refers to the ideal situation that the transport will not be impacted by reducing the number of ACKs. "Actual Goodput" refers to the real situation that the transport performance is impacted by both the "negative effect" and the "positive effect" when reducing the number of ACKs.

Figure 13(a) shows that the goodput gain is enlarged over a faster wireless link. Figure 13(b) demonstrates that TACK's ideal goodput approaches the transport upper bound with a minimized ACK frequency.

We then compare the actual goodput of TCP-TACK flows and TCP BBR flows over the IEEE 802.11b/g/n/ac wireless links. A Wi-Fi host (Intel Wireless-AC 8260, 2 $\times$ 2) is connected to another wired host with a wireless router (TL-WDR7500) forwarding. All devices are in a public room with over 10 additional APs and over 100 wireless users at peak time. Ping test shows that the RTT varies between 4 to 200 ms and slight burst losses exist. Single-flow tests are repeatedly conducted over all hours of the days in a full week. Figure 14(a) shows that TCP-TACK obtains 20% $\sim$ 28.1% of average goodput improvement over TCP BBR. Our data traces also show that TCP-TACK sends much less number of ACKs than TCP BBR (*e.g.*, over the 802.11g wireless links, $\frac{number\ of\ ACKs}{number\ of\ data\ packets}$ of TCP-TACK approximates 1.9%, and

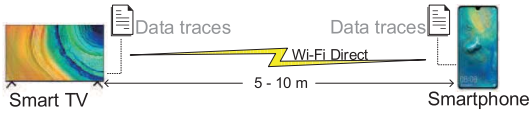


Fig. 15. Topology of wireless projection.

Metrics	RTP+UDP	TCP CUBIC	TCP BBR	TCP-TACK
Macroblocking (times/30min)	5 ~ 6	0	0	0
Rebuffering (%)	5 ~ 15	50 ~ 90	30 ~ 58	3 ~ 10
Number of ACKs (Order of magnitude)	0	6	6	4

Fig. 16. Performance of wireless projection with Miracast.

which of TCP BBR approximates 50%), significantly reducing the contentions on wireless links.

We also investigate the difference between the actual and the ideal goodput of TCP BBR with prior ACK thinning mechanisms. We check both scenarios with and without artificial packet loss (*i.e.*, $\rho = 0.1\%$ and $\rho = 0$, respectively). Figure 14(b) shows that legacy TCP's actual trend of goodput improvements does not match the ideal trend (as illustrated in Figure 13(b)). We believe it is because TCP's control algorithms such as loss recovery, round-trip timing, and send rate control are impacted by reducing ACK frequency. In contrast, TCP-TACK's actual performance approaches the ideal goodput improvement. This validates the TACK-based protocol design. We have also tested the Wi-Fi Direct [72] links, the results of which remain similar.

D. Deployment Experience: Miracast

TCP-TACK is deployed in the commercial products, such as Huawei Mate20 Series Smartphone (Android 9) [73] and Honor Smart TV [74], providing optimized high resolution wireless projection using Miracast. Miracast [45] allows users to wirelessly share multimedia, including high-resolution pictures and HD video content between Wi-Fi devices. A predecessor (Android 8) of Huawei's product adopts RTP on top of UDP as the transport protocol, while the current commercial products have modified Miracast so as to enable the TCP-based transmissions, *i.e.*, TCP CUBIC, TCP BBR, and TCP-TACK.

The first lesson we have learned during our deployment of Miracast is that BBR cannot perform well in two aspects. First, *ProbeRTT* only sends few packets every RTT, a sudden drop in throughput results in rebuffering of real-time video streaming. It is suggested that we remove the *ProbeRTT* directly, or just make it less drastic (*e.g.* 0.75x pacing rate) and more frequent (*e.g.*, every 2.5 s) as specified in [75].

Second, BBR converges slowly under bandwidth change. In [17], Neal Cardwell *et al.* elaborated the decrease and increase details for TCP BBR under bandwidth change. In that experiment, a single long-lived flow runs under 10 Mbps bandwidth with 40 ms RTT. At $t = 20$ s, the bandwidth is doubled to 20 Mbps. At $t = 40$ s, the bandwidth is halved to 10 Mbps. Since bandwidth estimate increases 1.95 times ($\approx 1.25^3$) in 3 *ProbeBW* cycles, TCP BBR takes about 960 ms (24 RTTs, $RTT = 40$ ms) to converge when the bandwidth is doubled. For the Miracast use case, it is recommended that we set the duration of ProbeBW as 16 TACK interval which approximates half of BBR's 8 RTTs. Similarly, the bandwidth filter window θ_{filter} is set 20 TACK intervals (approximates 5 RTTs). In the case of bandwidth change, TCP-TACK, however, only requires half of time (12 RTTs). On the other

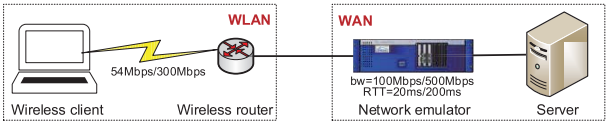


Fig. 17. Hybrid topology of WLAN and WAN.

Case	WLAN		WAN		TCP BBR			TCP-TACK		
	(ρ, ρ') (%)	bw (Mbps)	RTT (ms)	bw (Mbps)	Goodput (Mbps)	Data pkt (#)	ACK (#)	Goodput (Mbps)	Data pkt (#)	ACK (#)
1	(0, 0)	54	20	100	17.16	190896	104298	20.21	222561	24356
2	(1, 1)	54	20	100	16.90	175434	84523	18.44	213161	26068
3	(0, 0)	300	200	500	159.50	1657476	882545	190.22	2067212	2474
4	(1, 1)	300	200	500	156.39	1767197	897361	185.73	2204647	22407

Fig. 18. Performance over combined links of WLAN and WAN.

hand, the inflight drops from 200 KB down to 100 KB at $t = 42$ s due to the $inflight\_cap$ dropping from a 200 KB value adapted to 20 Mbps down to a 100KB value adapted to 10 Mbps. Since this duration is decided by the bandwidth filter window θ_{filter}, TCP BBR takes about 1600 ms (10 RTTs, $RTT = 160$ ms) and TCP-TACK only takes half of the time (5 RTTs). After $t = 42$ s, TCP-TACK has the same the behavior as TCP BBR.[3]

The topology of wireless projection with Miracast is illustrated in Figure 15. The smartphone screen can be projected to a nearby TV, wherein the distance is usually less than 10 meters between the two devices. Data traces are collected from both the smartphones and TVs during A/B testing. The table in Figure 16 summarizes the trace-based performance results. We found that TCP-TACK's video rebuffering ratio [46] is significantly reduced as compared with the legacy TCP or RTP based projections. Also TCP-TACK's macroblocking artifacts are less as compared with the RTP transport. The application-level benefit of TCP-TACK can be attributed to goodput improvement because of reduced ACK overhead and effective loss recovery. These experiences demonstrate TACK's significant advantages for high-throughput and reliable wireless transport.

E. Performance Over Combined Links of WLAN and WAN

TACK also works on the hybrid connections over both wired and wireless links. Figure 17 illustrates the topology. A wireless client (Intel Wireless-AC 8260, 2×2) connects a wireless router (TL-WDR7500) within a distance of 10 meters. Bandwidth of WLAN is configured by setting different 802.11 standards on the wireless router. For example, the policy of "802.11g only" provides a 54 Mbps bandwidth for WLAN. A network emulator is deployed to provide packet impairments and transport latency between the wireless router and a wired server. For example, setting the latency of 100 ms on both ingress and egress ports of network emulator provides a 200 ms RTT for the WAN. Packet loss rate on the data path (ρ) and on the ACK path (ρ') can also be set on the ingress port and egress port, respectively.

The table in Figure 18 shows the results when bandwidth of WLAN is the bottleneck. *Case 1* and *Case 2* consider the cases where a wireless client communicates with a domestic server. *Case 3* and *Case 4* consider the cases where a wireless client communicate with a cross-country server. All cases demonstrate TCP-TACK's advantage over legacy TCP on goodput. This can be attributed to two reasons: (1) ACK frequency reduction improves WLAN bandwidth utilization, and (2) TCP-TACK's advancements in loss recovery, round-trip timing and send rate control assure robust transmission over the long-delay and lossy links of WAN. We have also checked the scenario when the WAN is the bottleneck, no significant differences were observed between the two schemes.

Note that the number of ACKs of TCP-TACK in *Case 1* is nearly 10 times of that in *Case 3*, this is because the RTT on the WAN link has increased to 10 times in *Case 3*. According to Equation (4), the higher RTT results in the lower ACK frequency even though the data throughput is substantially higher. In addition, the number of ACKs in *Case 4* is nearly 20 K larger than that in *Case 3*, this is because TCP-TACK adds more ACKs on the ACK path when losses occur, in which the additional ACKs are almost IACKs.

Although the TACK-based protocols are designed for WLAN scenarios, in our previous conference paper [16], we also conducted experiments on TCP-TACK's performance in pure WAN scenarios. We summarize that TCP-TACK performs equally well as high-speed TCP variants because it removes the "negative effect" of reducing ACK frequency.

VIII. Discussion, Limitations, and Future Work

Buffer requirement. Sending fewer ACKs increases bottleneck buffer requirement. Ideally, buffer requirement is decided by the minimum send window (W_{min}), *i.e.*, $W_{min} - bdp$. Given by [10], we have $W_{min} = \frac{\beta}{\beta-1} \cdot bdp$, $\beta \geq 2$. By default, TCP-TACK ($\beta = 4$) requires a bottleneck buffer of 0.33 bdp. However, in practice, buffer requirement might be enlarged when the send rate control does not behave properly under network dynamics. Pacing can help alleviate the problems associated with increased buffer requirements [10], [61], [77], [78]. However, more substantial measurements are needed for a deep dive into the buffer requirement of TACK-based protocols in the future.

Handling reordering. Load balancing usually splits traffic across multiple paths at a fine granularity [79]–[82]. By handling the prevalent small degree of reordering on the transport layer [83], we help network layer to achieve fine partition granularity by enabling the load balancer to consider less about reordering avoidance in traffic engineering. Thus, we define the IACK delay as an allowance for settling time ([84] and [55] recommend $\frac{RTT_{min}}{4}$) before marking a packet lost. In general, the IACK delay depends on the service's tolerance of retransmission redundancy. It can be adjusted dynamically according to whether unnecessary retransmissions occur, which we leave as the further work.

Deployment issues. TCP-TACK depends on the middleboxes to permit the *extended-option-packets* through, which might limit applicable scenarios. Thus we acknowledge that TCP-TACK only works for a prototype implementation or in confined environments (*e.g.*, WLAN), a general deployment of TCP-TACK requires dealing with some issues where TCP options are modified or removed [85]. QUIC [34] is a flexible framework of transport protocol that uses UDP as a substrate to avoid requiring changes to legacy operating systems and middleboxes, and encrypts most of the packets including ACKs to avoid incurring a dependency on middleboxes. We consider an implementation of TACK upon QUIC as a future work.

[3]The $inflight\_cap$ reduction from roughly 100 KB to 50 KB after t=42 s is because that TCP BBR's actual send rate is computed to be 1% lower than the computed pacing rate [76]. TCP-TACK follows the same design to gradually drain any excess queue.

IX. Conclusion

In this paper, we revisited the acknowledgment mechanism for transport control, and we gave the detailed modeling, analysis, and implementation of a full protocol design with the minimized ACK frequency required on the transport layer. The TACK-based acknowledgment mechanism introduces more types of ACKs and carries more information in ACKs so as to reduce the number of ACKs required. In particular, IACKs speed up feedback for different instant events, and TACK periodically assures feedback robustness by carrying rich information in ACKs. The protocols based on TACK are therefore capable to achieve robust loss recovery, accurate round-trip timing, and effective send rate control. A TACK-based protocol is a good replacement of the legacy TCP to compensate for scenarios where the acknowledgment overhead is non-negligible.

Acknowledgment

The authors would like to thank the anonymous reviewers for their comments that have greatly helped to improve the manuscript.

References

[1] iLab. (2019). *Top 10 Traffic Killers Among Internet Videos*. [Online]. Available: https://www-file.huawei.com/-/media/corporate/pdf/whitepaper/10.pdf

[2] Cisco. (2020). *Cisco Visual Networking Index: Forecast and Trends, 2018–2023*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[3] International Standards Associations. (2016). *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. [Online]. Available: https://ieeexplore.ieee.org/document/7786995

[4] E. Magistretti, K. K. Chintalapudi, B. Radunovic, and R. Ramjee, "WiFi-nano: Reclaiming WiFi efficiency through 800 ns slots," in *Proc. ACM MobiCom*, 2011, pp. 37–48.

[5] L. Salameh, A. Zhushi, M. Handley, K. Jamieson, and B. Karp, "Hack: Hierarchical ACKs for efficient wireless medium utilization," in *Proc. USENIX ATC*, 2014, pp. 359–370.

[6] R. Braden, *Requirements for Internet Hosts—Communication Layers*, document RFC 1122, IETF, 1989.

[7] M. Allman, V. Paxson, and E. Blanton, *TCP Congestion Control*, document RFC 5681, IETF, 2009.

[8] R. de Oliveira and T. Braun, "A smart TCP acknowledgment approach for multihop wireless networks," *IEEE Trans. Mobile Comput.*, vol. 6, no. 2, pp. 192–205, Feb. 2007.

[9] S. Floyd and E. Kohler, *Profile for Datagram Congestion Control Protocol (DCCP)*, document RFC 4341, IETF, 2006.

[10] S. Landström and L.-Å. Larzon, "Reducing the TCP acknowledgment frequency," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, pp. 5–16, Jul. 2007.

[11] M. Gerla, K. Tang, and R. Bagrodia, "TCP performance in wireless multi-hop networks," in *Proc. IEEE WMCSA*, Feb. 1999, pp. 1–10.

[12] E. Altman and T. Jiménez, "Novel delayed ACK techniques for improving TCP performance in multihop wireless networks," in *Proc. IFIP PWC*, 2003, pp. 237–250.

[13] A. Bhartia *et al.*, "Measurement-based, practical techniques to improve 802.11 ac performance," in *Proc. IMC*, Nov. 2017, pp. 205–219.

[14] K. N. Rao, Y. K. S. Krishna, and K. N. Lakshminadh, "Improving TCP performance with delayed acknowledgments over wireless networks: A receiver side solution," in *Proc. IET Commun. Comput.*, Sep. 2013, pp. 195–201.

[15] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, 1988.

[16] T. Li *et al.*, "TACK: Improving wireless transport performance by taming acknowledgments," in *Proc. ACM SIGCOMM*, Jul. 2020, pp. 15–30.

[17] 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.

[18] H. Balakrishnan, V. N. Padmanabhan, G. Fairhurst, and M. Sooriyabandara, *TCP Performance Implications of Network Path Asymmetry*, document RFC 3449, IETF, 2002.

[19] F. Ge, L. Tan, and M. Zukerman, "Throughput of FAST TCP in asymmetric networks," *IEEE Commun. Lett.*, vol. 12, no. 2, pp. 158–160, Feb. 2008.

[20] C. P. Fu and S. C. Liew, "A remedy for performance degradation of TCP Vegas in asymmetric networks," *IEEE Commun. Lett.*, vol. 7, no. 1, pp. 42–44, Jan. 2003.

[21] C.-Y. Ho, C.-Y. Ho, and J.-T. Wang, "Performance improvement of delay-based TCPs in asymmetric networks," *IEEE Commun. Lett.*, vol. 15, no. 3, pp. 355–357, Mar. 2011.

[22] J. Park, D. Park, S. Hong, and J. Park, "Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing," *Comput. Commun.*, vol. 34, no. 6, pp. 730–742, May 2011.

[23] H. Chen, Z. Guo, R. Y. Yao, X. Shen, and Y. Li, "Performance analysis of delayed acknowledgment scheme in UWB-based high-rate WPAN," *IEEE Trans. Veh. Technol.*, vol. 55, no. 2, pp. 606–621, Mar. 2006.

[24] R. de Oliveira and T. Braun, "A dynamic adaptive acknowledgment strategy for TCP over multihop wireless networks," in *Proc. IEEE INFOCOM*, Mar. 2005, pp. 39–49.

[25] J. Chen, M. Gerla, Y. Z. Lee, and M. Y. Sanadidi, "TCP with delayed ACK for wireless networks," *Ad Hoc Netw.*, vol. 6, no. 7, pp. 1098–1116, Sep. 2008.

[26] F. R. Armaghani, S. S. Jamuar, S. Khatun, and M. F. A. Rasid, "Performance analysis of TCP with delayed acknowledgments in multi-hop ad-hoc networks," *Wireless Pers. Commun.*, vol. 56, no. 4, pp. 791–811, Feb. 2011.

[27] A. M. Al-Jubari, M. Othman, B. M. Ali, and N. A. W. A. Hamid, "An adaptive delayed acknowledgment strategy to improve TCP performance in multi-hop wireless networks," *Wireless Pers. Commun.*, vol. 69, no. 1, pp. 307–333, Mar. 2013.

[28] M. Allman, "On the generation and use of TCP acknowledgments," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 5, pp. 4–21, Oct. 1998.

[29] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "pHost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proc. ACM CONEXT*, Dec. 2015, pp. 1–12.

[30] L. Xu, K. Xu, Y. Jiang, F. Ren, and H. Wang, "Throughput optimization of TCP incast congestion control in large-scale datacenter networks," *Comput. Netw.*, vol. 124, pp. 46–60, Sep. 2017.

[31] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM*, Aug. 2017, pp. 239–252.

[32] M. Handley *et al.*, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. ACM SIGCOMM*, Aug. 2017, pp. 29–42.

[33] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. ACM SIGCOMM*, Aug. 2018, pp. 221–235.

[34] A. Langley *et al.*, "The QUIC transport protocol: Design and internet-scale deployment," in *Proc. ACM SIGCOMM*, Aug. 2017, pp. 183–196.

[35] J. Iyengar and I. Swett, *QUIC Loss Recovery and Congestion Control*, document draft 30, IETF, 2020.

[36] V. Paxson *et al.*, *Known TCP Implementation Problems*, document RFC 2525, IETF, 1999.

[37] G. Fairhurst, A. Custura, and T. Jones, *Changing the Default QUIC ACK Policy*, document draft 03, IETF, 2020.

[38] N. Kuhn, G. Fairhurst, J. Border, and E. Stephan, *QUIC for SATCOM*, document draft 06, IETF, 2020.

[39] J. Iyengar and I. Swett, *Sender Control of Acknowledgement Delays in QUIC*, document draft 02, IETF, 2020.

[40] T. Li, K. Zheng, R. Jadhav, and J. Kang, *Optimizing ACK Mechanism for QUIC*, document draft 00, IETF, 2020.

[41] T. Li, K. Zheng, K. Xu, and Y. Cui, "Acknowledgment on demand for transport control," *IEEE Internet Comput.*, vol. 25, no. 2, pp. 109–115, Mar. 2021.

[42] Cisco. (2019). *Cisco Predicts More IP Traffic in the Next Five Years Than in the History of the Internet*. [Online]. Available: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1955935

[43] K. Xu, L. Lv, T. Li, M. Shen, H. Wang, and K. Yang, "Minimizing tardiness for data-intensive applications in heterogeneous systems: A matching theory perspective," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 1, pp. 144–158, Jan. 2020.

[44] K. Xu *et al.*, "Modeling, analysis, and implementation of universal acceleration platform across online video sharing sites," *IEEE Trans. Services Comput.*, vol. 11, no. 3, pp. 534–548, May/Jun. 2018.

[45] Wi-Fi-Alliance. (2019). *High-Definition Content Sharing on Wi-Fi Devices Everywhere*. [Online]. Available: https://www.wi-fi.org/discover-wi-fi/miracast

[46] F. Dobrian *et al.*, "Understanding the impact of video quality on user engagement," in *Proc. ACM SIGCOMM*, 2011, pp. 362–373.

[47] Fillpthepipe. (2019). *Ackemu*. [Online]. Available: https://github.com/fillthepipe/ackemu

[48] S. D. Strowes, "Passively measuring TCP round-trip times," *Commun. ACM*, vol. 56, no. 10, pp. 57–64, Oct. 2013.

[49] Google WebRTC Team. (2019). *WebRTC*. [Online]. Available: https://webrtc.org/

[50] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Comput. Netw.*, vol. 51, no. 7, pp. 1777–1799, 2007.

[51] E. He, J. Leigh, O. Yu, and T. A. DeFanti, "Reliable blast UDP: Predictable high performance bulk data transfer," in *Proc. IEEE Cluster Comput.*, Sep. 2002, p. 317.

[52] R. Fox, *TCP Big Window and NAK Options*, document RFC 1106, IETF, 1989.

[53] B. Adamson, C. Bormann, M. Handley, and J. Macker, *NACK-Oriented Reliable Multicast (NORM) Transport Protocol*, document RFC 5740, IETF, 2009.

[54] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, *TCP Selective Acknowledgment Options*, document RFC 2018, IETF, 1996.

[55] Y. Cheng and N. Cardwell, *RACK: A Time-Based Fast Loss Detection Algorithm for TCP*, document draft 15, IETF, 2016.

[56] V. Paxson, M. Allman, H. J. Chu, and M. Sargent, *Computing TCP's Retransmission Timer*, document RFC 6298, IETF, 2011.

[57] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. USENIX NSDI*, Jul. 2018, pp. 329–342.

[58] Pantheon. (2018). *Pantheon of Congestion Control*. [Online]. Available: http://pantheon.stanford.edu/

[59] Pantheon-Before. (2018). *Test From GCE Tokyo to GCE Sydney Before the Advanced Round-Trip Timing is Applied*. [Online]. Available: https://pantheon.stanford.edu/result/4623/

[60] Pantheon-After. (2018). *Test From GCE Tokyo to GCE Sydney After the Advanced Round-Trip Timing is Applied*. [Online]. Available: https://pantheon.stanford.edu/result/4874/

[61] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 1157–1165.

[62] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[63] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP VEGAS: New techniques for congestion detection and avoidance," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 24–35, 1994.

[64] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–12.

[65] W. S. Lai, "An analysis of piggybacking in packet networks," *Comput. Netw.*, vol. 6, no. 4, pp. 279–290, Sep. 1982.

[66] *Transmission Control Protocol*, document RFC 793, IETF, DARPA, 1981.

[67] N. Cardwell *et al.* (2018). *BBR IETF 101 Update*. [Online]. Available: https://datatracker.ietf.org/meeting/101/materials/slides-101-iccrg-an-update-on-bbr-work-at/-google-00

[68] Spirent. (2017). *Accurate and Repeatable Network Emulation*. [Online]. Available: https://www.spirent.com/Products/Attero

[69] L. Rizzo. (2019). *Netmap—The Fast Packet I/O Framework*. [Online]. Available: http://info.iet.unipi.it/~luigi/netmap/

[70] Fillpthepipe. (2020). *A Patch to Allow Changing TCP ACK Frequency*. [Online]. Available: https://github.com/fillthepipe/TcpAckThinning

[71] Linux Man-Pages Project. (2020). *BPF Helpers*. [Online]. Available: http://man7.org/linux/man-pages/man7/bpf-helpers.7.html

[72] Wi-Fi-Alliance. (2019). *Wi-Fi Direct*. [Online]. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[73] Huawei. (2018). *Mate 20 Series Wireless Projection*. [Online]. Available: https://consumer.huawei.com/en/support/content/en-us00677996/

[74] Honor. (2019). *Honor Smart Screen*. [Online]. Available: https://consumer.huawei.com/en/support/content/en-us00677996/

[75] N. Cardwell *et al.* (2018). *BBR IETF 102 Update*. [Online]. Available: https://datatracker.ietf.org/meeting/102/materials/slides-102-iccrg-an-update-on-bbr-work-at-google-00

[76] N. Cardwell. (2020). *How to Make Inflight to Convergency 1 BDP When Inflight is 2 BDP Currently*. [Online]. Available: https://groups.google.com/g/bbr-dev/c/50AQUVQaGCg/m/SSZUt90WAwAJ

[77] F. Zhang *et al.*, "TADOC: Text analytics directly on compression," *VLDB J.*, vol. 30, no. 2, pp. 163–188, Mar. 2021.

[78] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and W. Chen, "Efficient document analytics on compressed data: Method, challenges, algorithms, insights," *Proc. VLDB Endowment*, vol. 11, no. 11, pp. 1522–1535, Jul. 2018.

[79] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, 2007.

[80] T. Li, K. Wang, K. Xu, K. Yang, C. S. Magurawalage, and H. Wang, "Communication and computation cooperation in cloud radio access network with mobile edge computing," *CCF Trans. Netw.*, vol. 2, no. 1, pp. 43–56, Jun. 2019.

[81] F. Zhang, J. Zhai, B. He, S. Zhang, and W. Chen, "Understanding co-running behaviors on integrated CPU/GPU architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 905–918, Mar. 2017.

[82] F. Zhang, J. Zhai, B. Wu, B. He, and X. Du, "Automatic irregularity-aware fine-grained workload partitioning on integrated architectures," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 3, pp. 867–881, Mar. 2021.

[83] L. Li *et al.*, "A measurement study on multi-path TCP with multiple cellular carriers on high speed rails," in *Proc. ACM SIGCOMM*, Aug. 2018, pp. 161–175.

[84] S. Blanton, A. L. N. Reddy, M. Allman, and E. Blanton, *Improving the Robustness of TCP to Non-Congestion Events*, document RFC 4653, IETF, 2006.

[85] C. Raiciu *et al.*, "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proc. USENIX NSDI*, 2012, pp. 399–412.

Tong Li (Member, IEEE) received the B.E. degree from the School of Computer Science, Wuhan University, China, in 2012, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2017. He was a Visiting Scholar with the School of Computer Science and Electronic Engineering, University of Essex, U.K., in 2014 and 2016. His research interests include networking, distributed systems, and big data.

Kai Zheng (Senior Member, IEEE) is currently the Director of the Computer Network and Protocol Research Laboratory, Huawei Technologies. His research interests include architectures and protocols for the next generation networks, such as 5G/IoT networks, cloud oriented data center networks, RDMA networks, and real-time multimedia networks.

Ke Xu (Senior Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is currently a Full Professor with Tsinghua University. He has published more than 200 technical articles and holds 11 U.S. patents in the research areas of next-generation internet, blockchain systems, the Internet of Things, and network security. He is a member of ACM. He has guest-edited several special issues in IEEE and Springer Journals. He served as the Steering Committee Chair for IEEE/ACM IWQoS. He is an Editor of IEEE INTERNET OF THINGS JOURNAL.

Rahul Arvind Jadhav graduated in computer science from Mumbai University in 2001. Since 2001, he has been part of few startups where he saw projects from conception to market deployment. In 2009, he joined Huawei and had been part of 2012 Labs untill 2020. He is currently an Avid Coder and a System Engineer working on solutions involving network and transport optimization. He has contributed towards more than dozen open sources, including Linux Kernel. He has also contributed towards IETF protocol standardization in the domain. He has special interest in scalable mesh network architectures for low-power networks.

Tao Xiong received the B.E. degree in computing science from China University of Geoscience, China, in 2003, the M.E. degree in computer science and engineering from the University of New South Wales at Sydney, Sydney, NSW, Australia, in 2008, and the Ph.D. degree from the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, in 2014. His research interests include networking, cloud computing, high performance network protocol, and wireless networks.

Keith Winstein is currently an Assistant Professor of computer science and, by courtesy, of electrical engineering at Stanford University.

Kun Tan is currently the Vice President of CSI, heading the Distributed and Parallel Software Laboratory, Huawei. He has been working on various aspects in networking and networked systems, AI/serverless frameworks, and cloud computing. Before joining Huawei, he was a Senior Researcher/Research Manager at Microsoft Research Asia. He has published over 100 papers in top conferences and journals. He received the USENIX Test-of-Time Award in 2019.

APPENDIX A
NECESSITY OF CARRYING MORE INFORMATION IN TACK

MSS is the maximum segment size (MSS), bw is the throughput on the data path, and L indicates the number of full-sized data packets counted before sending an ACK. RTT_{min} is the minimum RTT observed over a long period of time, and β indicates the number of ACKs per RTT_{min}. Then we compute the frequency of TACK as follow:

$$f_{tack} = \min\{\frac{bw}{L \cdot MSS}, \frac{\beta}{RTT_{min}}\} \tag{1}$$

We use IACKs to report the most recent range of lost packets, with which the sender can retransmit lost packets timely upon IACK arrivals. Since IACKs might also be lost when there exist losses on the ACK path, TACKs are adopted to report the blocks of lost packets with the smallest serial numbers as the so-called "unacked list". We use ρ and ρ' to denote the loss rate on the data path and on the ACK path, respectively. Q denotes the primary number of blocks in the "unacked list" that a TACK has reported. It is easy to see that if $\rho' = 0$, then we can set $Q = 0$. However, when ρ' is large, the provisioning of Q might fail to meet the needs of loss recovery. In this section, we derive under what ρ' it is more profitable to use a TACK carrying more information.

A.1 When bdp is large

To ensure efficient loss recovery, during the time period of $\triangle t$, our goal is to employ the TACK to repeat all the blocks of lost packets that have been reported by the lost IACKs, that is, the number of lost IACKs should not exceed Q.

Considering the worst case in which there are no back-to-back packet losses, that is, each loss forms a "hole" in the receiver's buffer. According to Equation (1), when $bdp \geq \beta \cdot L \cdot MSS$, the receiver sends β ($\beta \geq 1$) TACKs every RTT. The maximum number of IACKs can be computed as $\rho \cdot \frac{bdp}{MSS}$, where $\triangle t = RTT$, and the number of lost IACKs is computed as $\rho \cdot \rho' \cdot \frac{bdp}{MSS}$ under an ACK loss rate of ρ'. Since the number of lost IACKs should not exceed Q, *i.e.*, $\rho \cdot \rho' \cdot \frac{bdp}{MSS} \leq Q$, we have

$$\rho' \leq \frac{Q \cdot MSS}{\rho \cdot bdp} \tag{2}$$

In this case, when $\rho' > \frac{Q \cdot MSS}{\rho \cdot bdp}$, it is more profitable to use a TACK carrying more information. And the additional number of blocks ($\triangle Q$) in the "unacked list" that the TACK should report is given by $\triangle Q = \frac{\rho \cdot \rho' \cdot bdp}{MSS} - Q$.

A.2 When bdp is small

According to Equation (1), when $bdp < \beta \cdot L \cdot MSS$, the TACK frequency is $f_{tack} = \frac{bw}{L \cdot MSS}$, where L is the number of full-sized data packets counted before sending an ACK.

During the time period of $\triangle t$, and the number of lost IACKs is computed as $\rho \cdot \rho' \cdot \frac{bw}{MSS} \cdot \triangle t$. And meanwhile, at least one TACK should be sent, *i.e.*, $\frac{bw}{L \cdot MSS} \cdot \triangle t = 1$. Since the number of lost IACKs should not exceed Q, *i.e.*, $\rho \cdot \rho' \cdot \frac{bw}{MSS} \cdot \frac{L \cdot MSS}{bw} \leq Q$, we have

$$\rho' \leq \frac{Q}{\rho \cdot L} \tag{3}$$

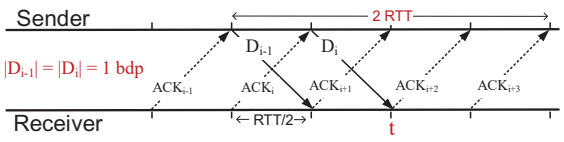


Fig. 1. Behavior analysis when $\beta = 2$.

In this case, when $\rho' > \frac{Q}{\rho \cdot L}$, it is more profitable to use a TACK carrying more information. And the additional number of blocks ($\triangle Q$) in the "unacked list" that the TACK should report is given by $\triangle Q = \frac{Q}{\rho \cdot L} - Q$.

To summarize, it can be derived that the rich information should be carried when the loss rate (ρ') on the ACK path follows:

$$\rho' > \begin{cases} \frac{Q \cdot MSS}{\rho \cdot bdp}, & bdp \geq \beta \cdot L \cdot MSS \\ \frac{Q}{\rho \cdot L}, & bdp < \beta \cdot L \cdot MSS \end{cases} \tag{4}$$

APPENDIX B
TACK FREQUENCY MINIMIZATION

TACK's frequency follows Equation (1), where β indicates the number of ACKs per RTT, and L indicates the number of full-sized data packets counted before sending an ACK. To minimize the ACK frequency, a smaller β or a larger L is expected. This section discusses the lower bound of β and the upper bound of L. We also give the default values suggested in practical scenarios. Finally, three insights are obtained through quantitatively analysis of TACK frequency.

B.1 Lower bound of β

With regard to the sliding-window protocols such as TCP, sending one ACK per RTT (*i.e.*, $\beta = 1$) transforms the protocol into a stop-and-wait mode. That is, the sender stops after sending a send window of data, and then waits for one RTT, *i.e.*, the time it takes for an ACK to reach the sender and the data released by this ACK to propagate to the receiver.

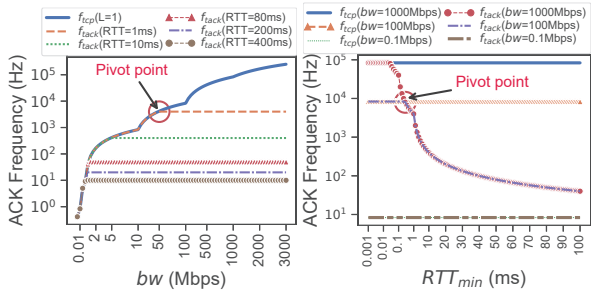
Since the waiting time wastes opportunities of sending data, a transport with $\beta = 1$ suffers from bandwidth under-utilization. Under these circumstances, two ACKs per RTT (*i.e.*, $\beta = 2$) are required. To facilitate the analysis, we assume that a symmetric network without loss. D_i denotes the data packets released by the i^{th} ACK (ACK_i) and $|D_i|$ denotes the data volume of D_i. As shown in Figure 1, to fully utilize the available bandwidth, at time t, the first byte of D_i should arrive at the receiver, and meanwhile ACK_{i+2} should acknowledge the last byte of D_{i-1}. Upon each ACK arrival, the sender will be enabled to send a bdp of data, *i.e.*, $|D_i| = bdp$. As a result, the send window size is $|D_i| + |D_{i-1}| = 2bdp$ and it takes 2 RTTs for the data in this window to complete. Note that the bottleneck buffer therefore has to be at least one bdp. In summary, the lower bound of β is 2.

B.2 Upper bound of L

According to Equation (3), we have

$$L \leq \frac{Q}{\rho \cdot \rho'} \tag{5}$$

Hence, the upper bound of L is given by $L = \frac{Q}{\rho \cdot \rho'}$. For example, when $Q = 4$, $\rho = \rho' = 10\%$, the receiver should send an ACK at least every $L = 400$ full-sized data packets.



(a) ACK frequency vs. bw (b) ACK frequency vs. RTT_{min}

Fig. 2. An example of ACK frequency dynamics (data packets are full-sized, $L = 1$, $\beta = 4$, and MSS = 1500 bytes).

B.3 Robustness consideration in TACK

According to Equation (1), the parameter β comes into effect when the bdp is large, and parameter L comes into effect when the bdp is small.

In terms of a transport with a large bdp, $\beta = 2$ should be sufficient to ensure utilization, but the large bottleneck buffer (*i.e.*, one bdp) makes it necessary to acknowledge data more often. In general, the minimum send window W_{min} can be roughly estimated as given in [1]:

$$W_{min} = \frac{\beta}{\beta - 1} \cdot bdp, \ \beta \geq 2 \tag{6}$$

Ideally, the bottleneck buffer requirement is decided by the minimum send window, *i.e.*, $W_{min} - bdp$. Since doubling the ACK frequency reduces the bottleneck buffer requirement substantially from one bdp to 0.33 bdp, this paper suggests $\beta = 4$ to provide redundancy, being more robust in practice.

Having a relatively low throughput, latency-sensitive flows (such as RPCs) and application-limited flows usually suffer more from ACK reduction as L grows. Since the high ACK frequency is not the main bottleneck in these cases, this paper suggests a delayed TCP-like provisioning of $L = 2$ to be more robust in practice. Note that we might also provide an option similar to `TCP_QUICKACK`, allowing the real-time applications to set $L = 1$.

B.4 Quantitatively analysis of TACK frequency

In this section, we give a quantitatively analysis of ACK frequency of TCP ($L = 1$) and TACK ($L = 1$). In the case that data packets are full-sized, we get three insights as follows.

First, given an L, the frequency of TACK is always no more than that of the legacy TCP ACK, *i.e.*, $f_{tack} \leq f_{tcp}$. For example as shown in Figure 2 , the frequency of TACK is only 10% of the per-packet ACK when bw = 48 Mbps and RTT_{min} = 10 ms, which is a typical scenario in WLAN.

Second, the higher bit rate over wireless links, the more number of ACKs are reduced by applying TACK. For example, the frequency of TACK has dropped two orders of magnitude ($f_{tack} \approx 2.4\% f_{tcp}$) when bw increases from 48 Mbps to 200 Mbps (RTT_{min} = 10 ms). Also, with higher bw, the RTT_{min} pivot point where the ACK frequency is reduced, is further lowered (Figure 2(a)).

Meanwhile, the larger latency between endpoints, the more number of ACKs are reduced by applying TACK. For example, the frequency of TACK has dropped three orders of magnitude ($f_{tack} \approx 0.3\% f_{tcp}$) when RTT_{min} increases from 10 ms to 80 ms (bw = 200 Mbps). And with larger RTT_{min}, the bw pivot point where the ACK frequency is reduced, is further lowered (Figure 2(b)).

In summary, TACK significantly reduces the ACK frequency in most cases. It is also straightforward that the results remain similar in the case that the data packets are not full-sized.

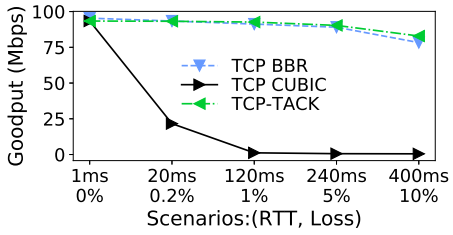
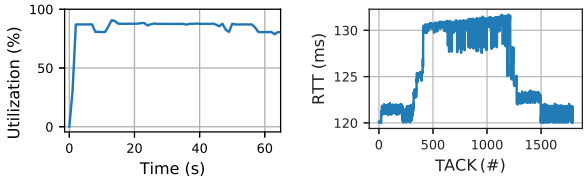
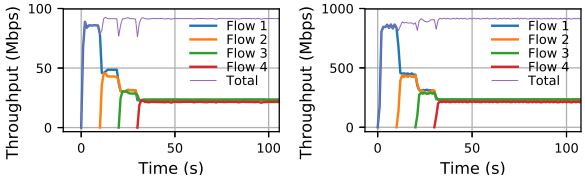


Fig. 3. Goodput vs. (RTT, Loss).



(a) Utilization (b) RTT

Fig. 4. Behavior under traffic change.



(a) Large buffer (b) Small buffer

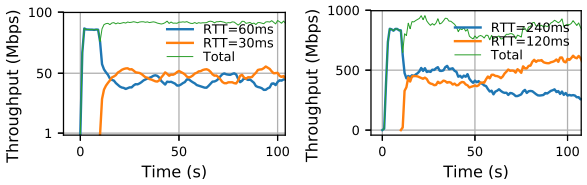
Fig. 5. Four flow convergence.

APPENDIX C
RECEIVER-BASED BBR EVALUATION

In this section, we conduct experiments to demonstrate that the receiver-based BBR in the context of TACK performs similar to the legacy BBR. Unless otherwise noted, TCP-TACK refers to the TCP-TACK that implements the receiver-based BBR.

Robustness over long-delay and lossy links. Figure 3 presents the average goodput achieved under various network delays and packet drops when transmitting a long-lived flow. The topology is formed by two hosts connected with a 100 Mbps wired link. Packet impairments are provided by the hardware network emulator between the end-hosts. It can be observed that both TCP BBR and TCP-TACK obtain a considerable goodput under long-delay and lossy network conditions, however, TCP CUBIC's goodput falls off rapidly with the increase of loss rate due to its loss-based congestion controller. TCP-TACK's efficiency of adaption to long-delay and lossy links can be attributed to two aspects. (1) TCP-TACK employs the receiver-based BBR to be robust to stochastic packet loss, and (2) the TACK-based protocol achieves effective loss recovery.

Traffic change adaptation. We start a long-lived flow under 1 Gbps bandwidth with 120 ms RTT. At t = 10 s, we start



(a) Large buffer (b) Small buffer

Fig. 6. RTT fairness.

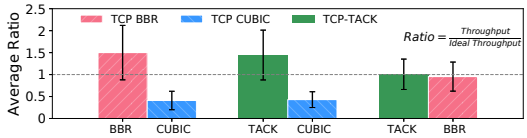


Fig. 7. TCP Friendliness.

3 new flows and let them stabilize for some time, then stop these 3 flows, leaving the original single flow in the system. Figure 4(a) shows that TCP-TACK absorbs the new burst of flows without disturbing the bandwidth utilization. Figure 4(b) shows timing detection in TCP-TACK absorbs the new traffic and drains afterwards.

Convergence dynamics. In this experiment, four long-lived flows share a bottleneck with 6 MB buffer. This bottleneck buffer is regarded as *large* under 100 Mbps bandwidth with 60 ms RTT, and *small* under 1 Gbps bandwidth with 120 ms RTT. The flows start their transfers ten seconds apart at 0, 10, 20, 30 s. Figure 5 shows that whenever a new flow starts, min-max fairness is maintained by TCP-TACK's adaptive rate controller. It also reveals that this reallocation is achieved without decreasing the utilization.

RTT Fairness. We start one flow at 0 s and another at 10 s with different RTTs. Figure 6(a) demonstrates that TCP-TACK converges smoothly to high utilization and fair bandwidth allocation when the bottleneck buffer is large. While in the small-buffer case as shown in Figure 6(b), the fair share shows randomness, which is similar to legacy BBR [2].

TCP Friendliness. We randomly sample throughput between 1 and 100 Mbps, RTT between 1 and 200 ms, buffer size between 0.5 and 5 *bdp*. The flows are run concurrently for 60 seconds. We report the average ratio of the throughput achieved by each flow to its ideal fair share for both the algorithm being tested. As shown in Figure 7, since TCP-TACK adopts a receiver-based BBR, it has a similar behavior [3] with legacy BBR in TCP friendliness.

References

[1] S. Landström and L.-A. Larzon, "Reducing the tcp acknowledgment frequency," *ACM SIGCOMM CCR*, vol. 37, no. 3, pp. 5–16, 2007.

[2] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of bbr congestion control," in *IEEE ICNP*, 2017, pp. 1–10.

[3] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR congestion control," https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-02.pdf, 2017.