TACK: Improving Wireless Transport Performance by Taming Acknowledgments

Tong Li
Huawei
li.tong@huawei.com

Kai Zheng
Huawei
kai.zheng@huawei.com

Ke Xu
Tsinghua University &
BNRist & PCL
xuke@tsinghua.edu.cn

Rahul Arvind Jadhav
Huawei
nyrahul@gmail.com

Tao Xiong
Huawei
yahz81@gmail.com

Keith Winstein
Stanford University
keithw@cs.stanford.edu

Kun Tan
Huawei
kun.tan@huawei.com

ABSTRACT

The shared nature of the wireless medium induces contention between data transport and backward signaling, such as acknowledgement. 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 acknowledgement called TACK ("Tame ACK"), as well as its TCP implementation TCP-TACK. 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. To minimize ACK frequency, TACK abandons the legacy *received-packet-driven* ACK. Instead, it balances byte-counting ACK and periodic ACK so as to achieve a controlled ACK frequency. 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 $\sim 28\%$ on goodput. We further find it performs equally well as high-speed TCP variants in wide area network (WAN) scenarios, this is attributed to the advancements of the TACK-based protocol design in loss recovery, round-trip timing, and send rate control.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Wireless local area networks**.

SIGCOMM '20, August 10–14, 2020, Virtual Event, NY, USA

ACM ISBN 978-1-4503-7955-7/20/08…$15.00
https://doi.org/10.1145/3387514.3405850

KEYWORDS

acknowledgement mechanism, ACK frequency, periodic ACK, instant ACK

ACM Reference Format:
Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. 2020. TACK: Improving Wireless Transport Performance by Taming Acknowledgments. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3387514.3405850

1 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 [45]. However, the average WLAN connection speed worldwide (*e.g.*, 30.3 Mbps in 2018 and predicted to be 92 Mbps by 2023 [24]) is far from satisfactory for these UHD-video-based applications (see §3.1).

It is well-studied that medium acquisition overhead in WLAN based on the IEEE 802.11 medium access control (MAC) protocol [11] can severely hamper TCP throughput, and TCP's many small ACKs are one reason [53, 69]. Basically, TCP sends an ACK for every one or two packets (*i.e.*, received-packet-driven) [7, 15]. ACKs share the same medium route with data packets, causing similar medium access overhead despite the much smaller size of the ACKs [8, 31, 36, 50, 58]. Contentions and collisions, as well as the wasted wireless resources by ACKs, lead to significant throughput decline on the data path (see §3.2).

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. Consequently, rethinking the way of TCP acknowledgement that reduces medium acquisition overhead in WLAN, so as to improve transport throughput, would be a relevant contribution.

The ACK frequency can be decreased by sending an ACK for every L ($L \geq 2$) incoming packets [8, 31, 50, 67] (*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 disturbs the packet clocking algorithms (*e.g.*, send pattern, send window update and loss detection) and round-trip timing [47], 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 §4.3).

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 acknowledgement 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 §4.4).

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 §5).

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

	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%

Figure 1: Percentage of goodput improvement of TCP-TACK over TCP-BBR in WLAN. Full results are in §6.3.

2 RELATED WORK

Reducing ACK frequency. In order to improve transport performance over IEEE 802.11 wireless links, Salameh *et al.* [69] 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 [13, 28, 34, 42, 64]. (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 [8], followed by a line of ACK thinning technologies [3, 9, 19, 20, 31, 57, 58, 67] 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). Both fail to match the number of ACKs to the frequency that a transport required in the network with time-varying data rate. 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 [6] 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.* [50] 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 [35], RCC [75], ExpressPass [22], NDP [40] and Homa [56] in datacenter environments.

TACK vs delayed ACK. Transport protocols, such as TCP and QUIC [51], also alternatively adopt delayed ACK [7, 15, 46]. 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 per-packet ACK when bw is small (see Equation (5)). TACK differs from delayed ACK by mandatorily sending ACKs periodically when bw is large (see Equation (3)). In particular, TACK applies periodic ACK when 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, for example the advancements in round-timing (see §5.2), are also applicable to TCP's delayed ACK scheme in the case, called "stretch ACK violation" [66], where ACKs are excessively delayed.

3 MOTIVATION

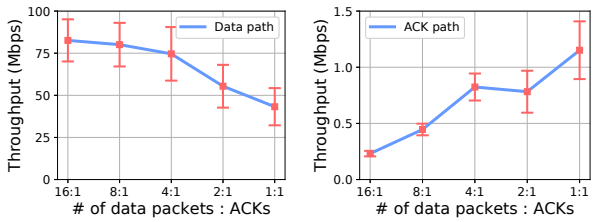
3.1 WLAN demands high throughput

It is predicted that, by 2022, the video-based applications will make up 82% of all IP traffic [23]. It is also reported that the video effect on the traffic is mainly because of the introduction of UHD video streaming [24]. As illustrated in 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.

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

Figure 2: Average bit rate of applications [24].

It is reported that [24] the average WLAN connection speed in 2018 was 30.3 Mbps and will be more than triple (92 Mbps) by 2023. 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 [45]). Wireless projection is a representative UHD-video-based application. A smartphone connects a TV using Wi-Fi Direct and streams videos on top of Miracast [4]. Our deployment experiences (see Figure 11) show that UDP-based solution



(a) Data throughput (b) ACK throughput

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

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* [25] due to bandwidth under-utilization. Reliable and high-throughput transport over WLAN turns out to be a challenging requirement.

3.2 Legacy 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 [53, 69]. 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 [7, 15], 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 [53, 69], 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 conducted emulations over the 802.11n wireless links with a PHY rate of 300 Mbps (see Figure 7). 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 10(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 [29] 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 [7, 15].

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

4 DESIGN RATIONALE

4.1 ACK frequency breakdown

ACK frequency can be denoted by f with the unit of Hz, *i.e.*, the number of ACKs per second, which 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)) [8, 31, 50, 67]. The frequency of byte-counting ACK is proportional to data throughput bw:

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

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 (*i.e.*, received-packet-driven). 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

$$f_{pack} = \frac{1}{\alpha} \tag{2}$$

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

Tame ACK (TACK). To control ACK frequency in the context of network dynamics, this paper proposes TACK, balancing the above two ways to minimize ACK frequency. As a result, we set the TACK frequency as $f_{tack} = \min\{f_b, f_{pack}\}$. Through Equations (1) and (2), we have $f_{tack} = \min\{\frac{bw}{L \cdot MSS}, \frac{1}{\alpha}\}$.

In 2006, Floyd and Kohler [31] proposed a tunable transport control variant in which the minimum ACK frequency allowed is twice per send window (*i.e.*, per RTT). In 2007, Sara Landström *et al.* [50] has also demonstrated that, in theory, acknowledging data twice per send window should be sufficient to ensure utilization with some modifications to the legacy TCP. Doubling the acknowledgment frequency to four times per send window can produce good performance and it is more robust in practice. Based on this rationale, we set $\alpha = \frac{RTT_{min}}{\beta}$, which means sending β ACKs per RTT_{min}. RTT_{min} is the smallest RTT observed over a long period of time. As a consequence, the frequency of TACK is eventually given as follow:

$$f_{tack} = \min\{\frac{bw}{L \cdot MSS}, \frac{\beta}{RTT_{min}}\} \tag{3}$$

Qualitatively, TACK turns to periodic ACK when bandwidth-delay product (bdp) is large ($bdp \geq \beta \cdot L \cdot MSS$), and falls back to byte-counting ACK when bdp is small ($bdp < \beta \cdot L \cdot MSS$). 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∼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[1] (see Appendix B.3). We also *quantitatively* analyze the ACK frequency below.

4.2 TACK frequency analysis

To facilitate the analysis, we assume that every data packet is full-sized (*i.e.*, MSS). When the TCP socket option `TCP_QUICKACK` is enabled, the legacy TCP sends an ACK for every packet (*i.e.*, per-packet ACK). The frequency of per-packet ACK is computed as

$$f_{tcp} = f_{tcp(L=1)} = \frac{bw}{MSS} \tag{4}$$

TCP also alternatively adopts delayed ACK [7, 15, 46], in which a data receiver may delay sending an ACK response by a given time interval (γ) or for every L full-sized incoming packets. As described in RFC 1122 [15] and updated in RFC 5681 [7], L is strictly limited up to 2, and γ is tens to hundreds of milliseconds and varies in different Linux distributions. The frequency of delayed ACK is computed as

$$f_{tcp\_delayed} = f_{tcp(L=2)} = \begin{cases} \frac{bw}{MSS}, & 0 \leq bw < \frac{2MSS}{\gamma} \\ \frac{bw}{2MSS}, & bw \geq \frac{2MSS}{\gamma} \end{cases} \tag{5}$$

According to Equations (3), (4) and (5), we summarize three insights as follows. First, given an L, the frequency of TACK is always no more than that of legacy TCP ACK, *i.e.*, $f_{tack} \leq f_{tcp}$. Second, the higher bit rate over wireless links, the more number of ACKs are reduced by applying TACK. Meanwhile, the larger latency between endpoints, the more

[1] Our real product deployment under both WAN and WLAN scenarios serves as a validation of its practicability. If not for special needs, it is not recommended to change the values of β and L.

number of ACKs are reduced by applying TACK. More detailed discussion on ACK frequency is given in Appendix B.4.

4.3 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 [71]), 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 (3), 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 4 (a)), after that, the sender calculates an RTT sample upon receiving a TACK. For example in Figure 4 (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 4 (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

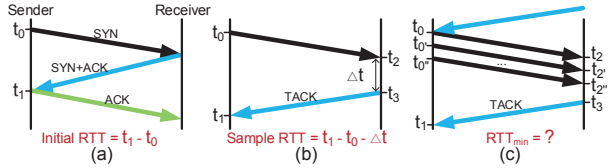


Figure 4: TACK-based round-trip timing: a case study.

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 announcement window (AWND). With a small frequency, TACK probably delays acknowledging packet receipts and reporting the AWND, 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 AWND = 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.

4.4 TACK-based acknowledgement 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 acknowledgement 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 acknowledgement 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 RTT_{min} 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 (§5.1). Specific and additional types of IACK can be defined on demand in the future.

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 §5.1). 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 (6)) 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 $\frac{\rho \cdot bw}{MSS}$. Since ρ is usually a small percentage (*e.g.*, $< 10\%$), this type of IACK only adds few number of ACKs on the return path.

5 TACK-BASED PROTOCOL DESIGN AND IMPLEMENTATION

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.

5.1 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 [46]. 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 byte). 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 retransmits $[1500 \sim 2999]$ with PKT.SEQ = 4. Assume the retransmitted 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 [51].

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. 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 5(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 [73], UDT [38], RBUDP [41], NACK Option [32], and NORM [1]). However, the proposed IACK in this paper is a novel concept of acknowledgement 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.

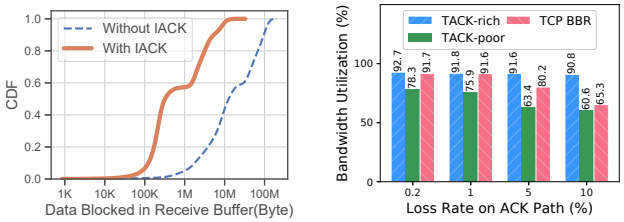
Specifically, the information carried in TACK contains ranges of packets which are *alternately* in the "*acked list*" and in the "*unacked list*". The "acked list" is a list of the blocks of contiguous packets that have been received and queued at the receiver, and the "unacked list" is a list of the 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 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:

$$\rho' > \begin{cases} \frac{Q \cdot MSS}{\rho \cdot bdp}, & bdp \geq \beta \cdot L \cdot MSS \\ \frac{Q}{\rho \cdot L}, & bdp < \beta \cdot L \cdot MSS \end{cases} \quad (6)$$

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 §6.1 for testbed setup). "TACK-poor" refers to the TACK-based TCP implementation (§5.4) 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 5(b) 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 [55] and RACK [21] 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 [55].



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

Figure 5: 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.

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.

5.2 Advancements in round-trip timing

As discussed above, legacy way of round-trip timing adopts simple RTT sampling (§4.3), 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 4 (c), a relative OWD sample can be computed as $OWD = t_2 - t_0$, where t_0 and t_2 are the packet

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

departure timestamp and the packet arrival timestamp, respectively. Upon packet arrivals, the receiver is capable to generate per-packet OWD samples.

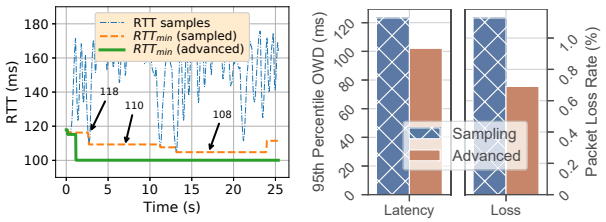
The smoothed OWD is an exponentially weighted moving average (EWMA) [65] of the per-packet OWD samples at the receiver. According to the smoothed OWD, the minimum OWD during each TACK interval can be observed. Afterwards, based on the TACK delay ($\Delta t^{\star}$) and the departure timestamp ($t_0^{\star}$) 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 [10, 17] 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 (§5.4) to transmit flows between two Wi-Fi endpoints, with a network emulator (see §6.1) forwarding. A fixed bidirectional latency (100 ms) is set between the endpoints. Figure 6(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 [59]. As illustrated in Figure 6(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 [60, 61]. We infer that an accurate minimum RTT estimate avoids pushing too much data into the pipe, and thus reduces latency and loss.

5.3 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 [2] 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 [39]), and *pacing_rate* may also be computed using bandwidth estimate of a rate-based controller (*e.g.*, BBR [17]).

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



(a) Microbenchmark (b) Wild Internet tests

Figure 6: 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 [61] and after [60] applying the advanced round-trip timing.

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 [16], and Compound TCP [72], a TACK-based congestion controller requires converting the CWND to the pacing rate [10]: $pacing\_rate_t \propto \frac{CWND}{sRTT_t}$, where $sRTT_t$ denotes the smoothed RTT at time t.

Most of the popularly used congestion controllers can work with the TACK design with minor implementation changes. Moreover, rate-based congestion controllers (*e.g.*, BBR) usually requires less changes than window-based ones (*e.g.*, CUBIC). In this paper, we adopt a TACK-based congestion controller co-designing BBR for TACK performance evaluation (§6). 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 acknowledgement 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.

In addition, lowering the ACK frequency also probably causes bandwidth under-utilization without timely updating the send window. To tackle this issue, an IACK updating the largest acknowledged packet and the AWND 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. Moreover, if a large volume of data[3] has been released in the receive buffer, an IACK may also be sent to update the AWND.

5.4 Protocol implementation

TACK, or its acknowledgement 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 §5.1$\sim$5.3. A full implementation of TCP-TACK including all the above advancements requires

[3]The trigger conditions of IACKs for updating send window can vary with different protocol implementations. An effective trigger should be carefully designed, which we leave to the further work.

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.

According to Equation (3), bw should be estimated in real time for TACK frequency update. This paper defines bw as the maximum delivery rate. The receiver computes the average delivery rate ($delivery\_rate$) per TACK interval as the ratio of data delivered to time elapsed. At time t, the maximum delivery rate is a windowed max-filtered value of the delivery rates, *i.e.*, $bw = \max(delivery\_rate_i)$, $\forall i \in [t - \theta_{filter}, t]$, where θ_{filter} is recommended as 5∼10 RTTs.

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. Note that bw and RTT_{min} are synced between sender and receiver, the sender therefore is capable to compute the number of expected TACKs based on the TACK frequency. IACKs are used for real-time synchronization in these cases.

6 EVALUATION

In this section, we first investigate the ideal performance of TACK over wireless links with various 802.11 standards. We then evaluate the actual performance of TCP-TACK in WLAN scenarios, including the deployment experience in commercial products. We also investigate how TCP-TACK would work over the combined links of WLAN and WAN. Finally, we share our long-term experience over WAN links, which further validates the advancements of the TACK-based protocol design.

6.1 Experiment setup

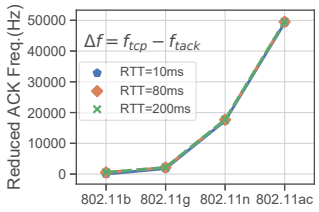
Experiment data is conducted on various wireless links (*e.g.*, IEEE 802.11b/g/n/ac), controllable links connected with a Spirent Attero network emulator [70], and shared links on the Internet, using the link conditions for randomized experimental trials. If not otherwise specified, the PHY raw bit rates of 802.11b/g/n/ac links are 11/54/300/866.7 Mbps, respectively. Detailed parameters are listed in Figure 7.

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

Figure 7: Parameters of 802.11-based links.

Since this paper mainly discusses acknowledgement 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 acknowledgement mechanisms in the context of the same congestion controller upon the same transport protocol. Particularly, TCP-TACK is compared with TCP BBR. TCP-TACK is implemented upon the TCP of our user-mode Stack[4]

[4]TCP-TACK has been widely applied in commercial products of Huawei. Because of Huawei's policies related to Linux's GNU GPL license, we currently only implement TCP-TACK in the user space instead of the kernel space. TCP-TACK evaluations in this paper are all conducted under this commercial implementation.



(a) ACK frequency reduction

	802.11b		802.11ac	
RTT_{min}	TCP (L=2)	TACK (L=2)	TCP (L=2)	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)

Figure 8: TACK reduces ACK frequency over the IEEE 802.11b/g/n/ac wireless links.

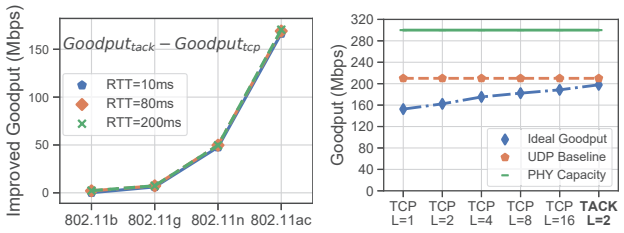
based on the Netmap framework [68]. Both TCP-TACK and TCP BBR has integrated the improvements specified in [18] to improve throughput over wireless links with aggregation.

The Linux kernel follows the TCP [7, 15] guidelines of sending an ACK for every second full-sized data packet received. For experimentation we changed the Linux Kernel 5.3 TCP code [30] 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 [54] (`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 [21] 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.

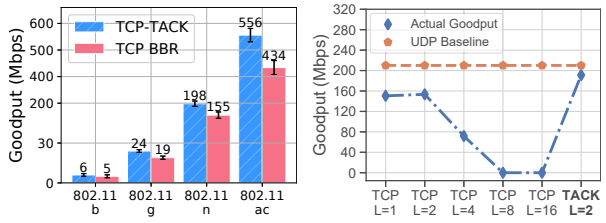
6.2 TACK frequency in real-world deployments

First of all, we give numeral analysis of TACK frequency over the 802.11 wireless links in comparison with standard delayed ACKs. Figure 8(a) shows that more number of ACKs are reduced in the case of a faster PHY capacity. Specifically, as shown in Figure 8(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 8(a) and 9(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.



(a) Goodput improvement (b) Ideal goodput trend

Figure 9: (a) Links with faster PHY rate enlarge goodput improvement. (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

Figure 10: (a) TCP-TACK obtains 20% ∼ 28.1% of goodput improvement. (b) Prior ACK thinning mechanisms disturb the TCP transport performance (RTT=80 ms, 802.11n).

6.3 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 [29] specified in §3.2 to estimate the ideal goodput of different ACK thinning techniques. For example, "TCP (L=8)" considers the case of byte-counting ACK that send an ACK every 8 full-sized packets. Our simulator keeps sending 1518-byte packets from the sender over the 802.11n links, and the receiver counts 8 received packets, and then sends one 64-byte packet as an ACK. "UDP Baseline" acts as the transport upper bound as its goodput is not disturbed by ACKs (see Figure 7). "PHY Capacity" is the raw bit rate at the PHY layer.

It is well-known that the transport will be disturbed 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 disturbed by reducing the number of ACKs. "Actual Goodput" refers to the real situation that the transport is impacted by both the "negative effect" and the "positive effect" when reducing the number of ACKs.

Figure 9(a) shows that the goodput gain is enlarged over a faster wireless link. Figure 9(b) demonstrates that TACK's ideal goodput approaches the transport upper bound with a minimized ACK frequency. Note that the gap between UDP baseline and PHY capacity is caused by non-ACK factors such as high medium acquisition overhead for data packets. Generating large A-MPDUs (Aggregated MAC Protocol Data Units [12]) may serve the purpose of reducing this gap. However, this is outside the scope of this paper.

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

Figure 11: Wireless projection with Miracast.

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 × 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 10(a) shows that TCP-TACK obtains 20% ∼ 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 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 introduce data packet impairments (ρ = 0.1%) by adding a network emulator connected between the endpoints. Figure 10(b) shows that legacy TCP's actual trend of goodput improvements does not match the ideal trend (as illustrated in Figure 9(b)). We believe it is because TCP's control algorithms such as loss recovery, round-trip timing, and send rate control are disturbed 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 [5] links, the results of which remain similar.

6.4 Deployment experience: Miracast

TCP-TACK is deployed in the commercial products, such as Huawei Mate20 Series Smartphone (Android 9) [43] and Honor Smart TV [44], providing optimized high resolution wireless projection using Miracast. Miracast [4] 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 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

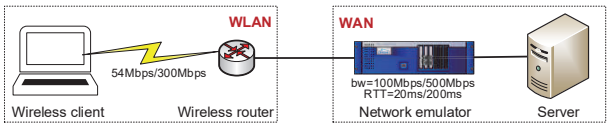


Figure 12: Hybrid topology of WLAN and WAN.

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

Figure 13: Performance over combined links of WLAN and WAN.

smartphones and TVs during A/B testing. Figure 11 summarizes the trace-based performance results. We found that TCP-TACK's video rebuffering ratio [25] 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.

6.5 Performance over combined links of WLAN and WAN

TACK also works on the hybrid connections over both wired and wireless links. Figure 12 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 the WLAN. A hardware 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.

Figure 13 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) Significant 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.

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 (3), the higher RTT results in the lower TACK 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 return path when losses occur, in which the additional ACKs are almost loss-event-driven IACKs.

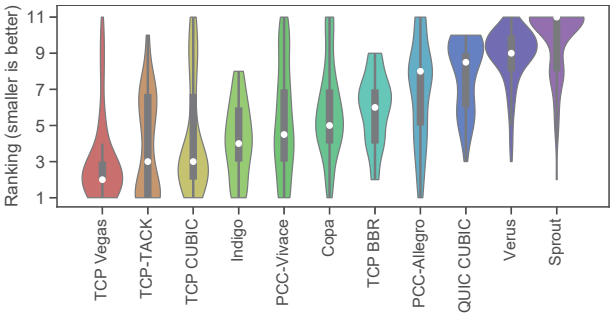


Figure 14: Violin-plots of performance ranking (200-day public data traces from [63]).

6.6 Experience over real-world WAN links

Although the TACK-based protocols are designed for WLAN scenarios, we tried evaluating the performance in WAN scenarios. Specifically, we have integrated TCP-TACK into the Pantheon [59] and run long-term tests in different workloads (single flow, or cross traffic). The Pantheon is a global-scale community evaluation platform for academic research on TCP variants. Measurement nodes are on wired networks and in cloud datacenters across nine countries such as the USA, the UK, Japan and Australia. Links between endpoints are with varying bandwidth and latency, and are non-dedicated, *i.e.*, there exists wild cross traffic over the Internet.

We investigate the overall performance of TCP-TACK as well as a curated collection of 10 working implementations of high-speed protocol variants: TCP Vegas [16], TCP CUBIC, TCP BBR [17], QUIC CUBIC (proto-quic [37]), PCC-Allegro [26], PCC-Vivace [27], Indigo [33], Copa [10], Verus [76], and Sprout [74]. All schemes are using their recommended parameters [62].

For quantitative analysis, we summarize the performance of each scheme using a version of Kleinrock's power metric [49] as the utility function $log(\frac{Throughput_{avg}}{OWD_{95th}})$, where $Throughput_{avg}$ denotes the mean throughput, and OWD_{95th} denotes the mean 95th percentile one-way delay (with clock synchronization [59]). Figure 14 illustrates the ranking of each scheme in tests during 200 days [63]. For each test, the highest-power scheme has the smallest ranking value according to the utility metric. Figure 14 reveals that TCP-TACK achieves acceptable performance in the WAN scenarios[5]. It is explained that the proposed TACK-based protocol design overcomes the hurdles for reducing the ACK frequency. Note that due to less contention between data packets and ACKs in WAN, TACK achieves less performance gain in WAN than that in WLAN.

[5]Note that QUIC CUBIC performs badly as it uses a very old version which has already been deprecated [37].

Our experience over real-world WAN links serves as a validation of the cooperation between TACK and receiver-based transport paradigm. However, we believe more substantial measurements are needed in the future, to answer the question of how TACK-based protocols behave on various real-world traffic patterns in WAN scenarios.

7 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 [50], 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 [2, 50]. Our experiment results in WAN scenarios (Figure 14) reveal that TCP-TACK's buffer requirement (based on OWD estimate) can be bounded in practice. 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 [48]. By handling the prevalent small degree of reordering on the transport layer [52], 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 ([14] and [21] 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.

Congestion controller. TACK impacts the *implementation* of congestion controllers. For example, to work with TACK, it is required to change sender-based control to receiver-based control. This paper adopted a TACK-based congestion controller co-designing BBR in a receiver-based way. Figure 15 shows that the co-designed BBR has the similar TCP friendliness to the standard BBR. In other words, as an ACK mechanism, our current experience shows that TACK does not impact much on the *performance* of congestion controllers. However, we believe more substantial investigations are needed in the future, to answer the question of how TACK works with more congestion controllers such as CUBIC, Vegas, and Compound TCP.

TCP splitting. TCP splitting is a possible way to reduce the complexity of the TACK-based protocol design. This is based on the fact that the last-mile wireless network usually has a smaller delay and converges fast. However, TCP splitting uses a proxy access node that divides the end-to-end TCP connection, which needs further modification on the access point (router). Another well-known problem of TCP splitting is that the split TCP connection is no longer reliable or secure, and a server failure may cause the client to believe that data has been successfully received when it has not. The cost performance of TACK with/without TCP splitting is worth being further studied.

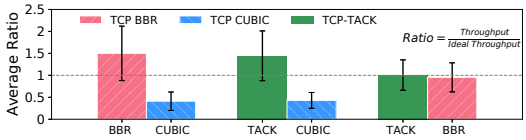


Figure 15: Evaluation of TCP friendliness. We randomly sample bandwidth between 1 and 100 Mbps, RTT between 1 and 200 ms, and bottleneck buffer size between 0.5 and 5 *bdp*. The flows are run concurrently for 60 seconds. We report the average ratio of the throughput (Y axis) achieved by each flow to its ideal fair share for all algorithms being tested.

8 CONCLUSION

To the best of our knowledge, this is the first work to give a full protocol design with minimized ACK frequency required on the transport layer. The TACK-based acknowledgement 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 (*e.g.*, packet losses), 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 acknowledgement overhead is non-negligible (*i.e.*, WLAN scenarios), and meanwhile, it also works well in WAN scenarios. This serves as a strong validation of TACK.

This work does not raise any ethical issues.

ACKNOWLEDGMENTS

We thank Yi Zeng, Xinping Chen, Shengjun Chen, and Ruixiang Guo from Huawei Computer Network and Protocol Lab for the work and support over the years. We thank Zhiqiang Fan, Hua Yu, Xiongzuo Pan, Dong Yang, Tao Bai, and Meng Luo for helping to build the testbed. We are grateful for conversations with and feedback from Li Li, Junsen Chen, Dang Su, Jian He, and Fanzhao Wang. We thank Feng Gao, Jing Zuo, and Fang Liu for helping to polish the language. We also thank the anonymous reviewers and our shepherd, Radhika Mittal, for their valuable feedback. Ke Xu is supported by NSFC Project with No. 61825204 and No. 61932016, Beijing Outstanding Young Scientist Program with No. BJJWZYJH01201910003011. Keith Winstein acknowledges funding from NSF grants CNS-1909212 and CNS-1763256 for the support on the Pantheon open-source project [33], which contributes to the horizontal evaluation of this work among a collection of transport protocols.

REFERENCES

[1] Brian Adamson, Carsten Bormann, Mark Handley, and Joe Macker. 2009. RFC 5740: Nack-oriented reliable multicast (NORM) transport protocol. *IETF* (2009).

[2] Amit Aggarwal, Stefan Savage, and Thomas Anderson. 2000. Understanding the performance of TCP pacing. In *Proceedings of IEEE INFOCOM*. 1157–1165.

[3] Ammar Mohammed Al-Jubari. 2013. An adaptive delayed acknowledgment strategy to improve TCP performance in multi-hop wireless networks. *Springer WPC* 69, 1 (2013), 307–333.

[4] Wi-Fi Alliance. 2019. High-definition content sharing on wi-fi devices everywhere. https://www.wi-fi.org/discover-wi-fi/miracast.

[5] Wi-Fi Alliance. 2019. Wi-Fi direct. https://www.wi-fi.org/discover-wi-fi/wi-fi-direct.

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

[7] Mark Allman, Vern Paxson, and Ethan Blanton. 2009. RFC 5681: TCP congestion control. *IETF* (2009).

[8] Eitan Altman and Tania Jiménez. 2003. Novel delayed ACK techniques for improving TCP performance in multihop wireless networks. In *Proceedings of IFIP PWC*. 237–250.

[9] Farzaneh Razavi Armaghani, Sudhanshu Shekhar Jamuar, Sabira Khatun, and Mohd Fadlee A. Rasid. 2011. Performance analysis of TCP with delayed acknowledgments in multi-hop ad-hoc networks. *Springer WPC* 56, 4 (2011), 791–811.

[10] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *Proceedings of USENIX NSDI*. 329–342.

[11] IEEE Standards Association. 2016. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. https://ieeexplore.ieee.org/document/7786995.

[12] Frank Aurzada, Martin Lévesque, Martin Maier, and Martin Reisslein. 2014. FiWi access networks based on next-generation PON and gigabit-class WLAN technologies: A capacity and delay analysis. *IEEE/ACM Transactions on Networking (ToN)* 22, 4 (2014), 1176–1189.

[13] Hari Balakrishnan, Venkata N. Padmanabhan, Godred Fairhurst, and Mahesh Sooriyabandara. 2002. RFC 3449: TCP performance implications of network path asymmetry. *IETF* (2002).

[14] Sumitha Blanton, A. L. Narasimha Reddy, Mark Allman, and Ethan Blanton. 2006. RFC 4653: Improving the robustness of TCP to non-congestion events. *IETF* (2006).

[15] R. Braden. 1989. RFC 1122: Requirements for internet hosts - communication layers. *IETF* (1989).

[16] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. 1994. TCP Vegas: New techniques for congestion detection and avoidance. *ACM SIGCOMM CCR* 24, 4 (1994), 24–35.

[17] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control. *ACM Queue* 14, 5 (2016), 20–53.

[18] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Ian Swett, Jana Iyengar, Victor Vasiliev, and Van Jacobson. 2018. BBR IETF 101 update. https://datatracker.ietf.org/meeting/101/materials/slides-101-iccrg-an-update-on-bbr-work-at/-google-00.

[19] Hongyuan Chen, Zihua Guo, Richard Yuqi Yao, Xuemin Shen, and Yanda Li. 2006. Performance analysis of delayed acknowledgment scheme in UWB-based high-rate WPAN. *IEEE TVT* 55, 2 (2006), 606–621.

[20] Jiwei Chen, Mario Gerla, Yeng Zhong Lee, and M. Y. Sanadidi. 2008. TCP with delayed ack for wireless networks. *Elsevier Ad Hoc Networks* 6, 7 (2008), 1098–1116.

[21] Yuchung Cheng and Neal Cardwell. 2016. RACK: A time-based fast loss detection algorithm for TCP. *Work in progress, IETF* (2016).

[22] Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-scheduled delay-bounded congestion control for datacenters. In *Proceedings of ACM SIGCOMM*. 239–252.

[23] Cisco. 2019. Cisco predicts more ip traffic in the next five years than in the history of the internet. https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1955935.

[24] Cisco. 2020. Cisco visual networking index: forecast and trends, 2018-2023. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[25] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Antony Joseph, Aditya Ganjam, Jibin Zhan, and Zhang Hui. 2011. Understanding the impact of video quality on user engagement. In *Proceedings of ACM SIGCOMM*.

[26] Mo Dong, Qingxi Li, Doron Zarchy, Philip Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting congestion control for consistent high performance. In *Proceedings of USENIX NSDI*. 395–408.

[27] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-learning congestion control. In *Proceedings of USENIX NSDI*. 343–356.

[28] Ge Fei, Liansheng Tan, and Moshe Zukerman. 2008. Throughput of FAST TCP in asymmetric networks. *IEEE Communications Letters* 12, 2 (2008), 158–160.

[29] Fillpthepipe. 2019. Ackemu. https://github.com/fillthepipe/ackemu.

[30] Fillpthepipe. 2020. A Patch to allow changing TCP ACK frequency. https://github.com/fillthepipe/TcpAckThinning.

[31] Sally Floyd and Eddie Kohler. 2006. RFC 4341: Profile for datagram congestion control protocol (DCCP). *IETF* (2006).

[32] Richard Fox. 1989. RFC 1106: TCP big window and nak options. *IETF* (1989).

[33] Y. Yan Francis, Ma Jestin, D. Hill Greg, Raghavan Deepti, S. Wahby Riad, Levis Philip, and Winstein Keith. 2018. Pantheon: The training ground for Internet congestion-control research. In *Proceedings of USENIX ATC*. 1–13.

[34] Cheng P. Fu and Soung C. Liew. 2003. A remedy for performance degradation of TCP vegas in asymmetric networks. *IEEE Communications Letters* 7, 1 (2003), 42–44.

[35] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. pHost: Distributed near-optimal datacenter transport over commodity network fabric. In *Proceedings of ACM CONEXT*.

[36] Mario Gerla, Ken Tang, and Rajive Bagrodia. 1999. TCP performance in wireless multi-hop networks. In *Proceedings of IEEE WMCSA*. 1–10.

[37] Google. 2019. Quic cubic implementation. https://github.com/google/proto-quic.

[38] Yunhong Gu and Robert L. Grossman. 2007. UDT: UDP-based data transfer for high-speed wide area networks. *Computer Networks* 51, 7 (2007), p.1777–1799.

[39] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008), 64–74.

[40] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wojcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of ACM SIGCOMM*. 29–42.

[41] Eric He, Jason Leigh, Oliver Yu, and Thomas A. DeFanti. 2002. Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. In *Proceedings of IEEE Cluster Computing*. 317.

[42] Cheng Yuan Ho, Cheng Yun Ho, and Jui Tang Wang. 2011. Performance improvement of delay-based TCPs in asymmetric networks. *IEEE Communications Letters* 15, 3 (2011), 355–357.

[43] Huawei. 2018. Mate 20 series wireless projection. https://consumer.huawei.com/en/support/content/en-us00677996/.

[44] Huawei. 2019. Honor smart screen. https://consumer.huawei.com/en/support/content/en-us00677996/.

[45] Huawei iLab. 2019. Top 10 traffic killers among Internet videos. https://www-file.huawei.com/-/media/corporate/pdf/whitepaper/10.pdf.

[46] Jana Iyengar and Ian Swett. 2020. QUIC loss recovery and congestion control. *IETF draft* (2020).

[47] Van Jacobson. 1988. Congestion avoidance and control. *ACM SIGCOMM CCR* 18, 4 (1988), 314–329.

[48] Srikanth Kandula, Dina Katabi, Arthur Berger, and Arthur Berger. 2007. Dynamic load balancing without packet reordering. *ACM SIGCOMM CCR* 37, 2 (2007), 51–62.

[49] Leonard Kleinrock. 1978. On flow control in computer networks. In *Proceedings of IEEE ICC*.

[50] Sara Landström and Lars-Ake Larzon. 2007. Reducing the TCP acknowledgment frequency. *ACM SIGCOMM CCR* 37, 3 (2007), 5–16.

[51] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade,

Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. 2017. The QUIC transport protocol: Design and internet-scale deployment. In *Proceedings of ACM SIGCOMM*. 183–196.
[52] Li Li, Ke Xu, Tong Li, Kai Zheng, Chunyi Peng, Dan Wang, Xiangxiang Wang, Meng Shen, and Rashid Mijumbi. 2018. A measurement study on multi-path TCP with multiple cellular carriers on high speed rails. In *Proceedings of ACM SIGCOMM*. 161–175.
[53] Eugenio Magistretti, Krishna Kant Chintalapudi, Bozidar Radunovic, and Ramachandran Ramjee. 2011. WiFi-Nano: Reclaiming WiFi efficiency through 800 ns slots. In *Proceedings of ACM MobiCom*. 37–48.
[54] Linux man-pages project. 2020. BPF helpers. http://man7.org/linux/man-pages/man7/bpf-helpers.7.html.
[55] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. 1996. RFC 2018: TCP selective acknowledgment options. *IETF* (1996).
[56] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of ACM SIGCOMM*.
[57] Ruy De Oliveira and Torsten Braun. 2005. A dynamic adaptive acknowledgment strategy for TCP over multihop wireless networks. In *Proceedings of IEEE INFOCOM*. 39–49.
[58] Ruy De Oliveira and Torsten Braun. 2006. A smart TCP acknowledgment approach for multihop wireless networks. *IEEE TMC* 6, 2 (2006), 192–205.
[59] Pantheon. 2018. Pantheon of congestion control. http://pantheon.stanford.edu/.
[60] Pantheon. 2018. Test from GCE Tokyo to GCE Sydney after the advanced round-trip timing is applied. https://pantheon.stanford.edu/result/4874/.
[61] Pantheon. 2018. Test from GCE Tokyo to GCE Sydney before the advanced round-trip timing is applied. https://pantheon.stanford.edu/result/4623/.
[62] Pantheon. 2019. Github repo of schemes tested in the pantheon. https://github.com/StanfordSNR/pantheon/tree/master/third_party.
[63] Pantheon. 2019. Summary of results in Pantheon. https://pantheon.stanford.edu/summary/.
[64] Jiyong Park, Daedong Park, Seongsoo Hong, and Jungkeun Park. 2011. Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing. *Elsevier Computer Communications* 34, 6 (2011), 730–742.
[65] Vern Paxson, Mark Allman, H.K. Jerry Chu, and Matt Sargent. 2011. RFC 6298: Computing TCP's retransmission timer. *IETF* (2011).
[66] Vern Paxson, Mark Allman, Scott Dawson, William C. Fenner, Jim Griner, Ian Heavens, Kevin Lahey, Jeff Semke, and Bernie Volz. 1999. RFC 2525: Known TCP implementation problems. *IETF* (1999).
[67] Kothuri Nageswara Rao, Y. K. Sundara Krishna, and K. Lakshminadh. 2013. Improving TCP performance with delayed acknowledgments over wireless networks: A receiver side solution. In *IET Communication and Computing*.
[68] Luigi Rizzo. 2019. Netmap - The fast packet i/o framework. http://info.iet.unipi.it/~luigi/netmap/.
[69] Lynne Salameh, Astrit Zhushi, Mark Handley, Kyle Jamieson, and Brad Karp. 2014. HACK: Hierarchical ACKs for efficient wireless medium utilization. In *Proceedings of USENIX ATC*. 359–370.
[70] Spirent. 2017. Accurate and repeatable network emulation. https://www.spirent.com/Products/Attero.
[71] Stephen D. Strowes. 2013. Passively measuring TCP round-trip times. *Commun. ACM* 56, 10 (2013), 57–64.
[72] Kun. Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. 2006. A compound TCP approach for high-speed and long distance networks. In *Proceedings of IEEE INFOCOM*. 1–12.
[73] Google Chrome team. 2019. WebRTC. https://webrtc.org/.
[74] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proceedings of USENIX NSDI*. 459–472.
[75] Lei Xu, Ke Xu, Yong Jiang, Fengyuan Ren, and Haiyang Wang. 2017. Throughput optimization of TCP incast congestion control in large-scale datacenter networks. *Elsevier Computer Networks* 124 (2017), 46–60.
[76] Yasir Zaki, Jay Chen, and Lakshminarayanan Subramanian. 2015. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of ACM SIGCOMM*. 509–522.

Appendices

Appendices are supporting material that has not been peer reviewed.

A NECESSITY OF CARRYING MORE INFORMATION IN TACK

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 (3), 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

$$\rho' \leq \frac{Q \cdot MSS}{\rho \cdot bdp} \tag{7}$$

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 (3), 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

$$\rho' \leq \frac{Q}{\rho \cdot L} \tag{8}$$

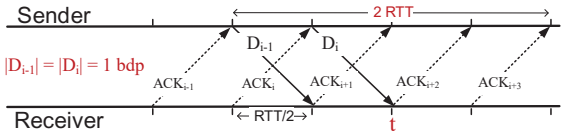


Figure 16: Behavior analysis when $\beta = 2$.

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:

$$\rho' > \begin{cases} \frac{Q \cdot MSS}{\rho \cdot bdp}, & bdp \geq \beta \cdot L \cdot MSS \\ \frac{Q}{\rho \cdot L}, & bdp < \beta \cdot L \cdot MSS \end{cases} \tag{9}$$

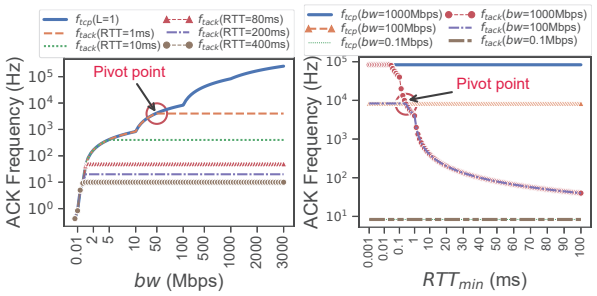
B TACK FREQUENCY MINIMIZATION

TACK's frequency follows Equation (3), 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 16, 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.



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

Figure 17: An example of ACK frequency dynamics (data packets are full-sized, $L = 1$ and $MSS = 1500$ bytes).

B.2 Upper bound of L

According to Equation (8), we have

$$L \leq \frac{Q}{\rho \cdot \rho'} \tag{10}$$

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.

B.3 Robustness consideration in TACK

According to Equation (3), 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 [50]:

$$W_{min} = \frac{\beta}{\beta - 1} \cdot bdp, \ \beta \geq 2 \tag{11}$$

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 ACK frequency modeling and analysis

In the case that data packets are full-sized, according to Equations (3), (4) and (5), 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 17 , 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 17(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 17(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.