TCPM Working Group
Internet-Draft
Intended status: Experimental
Expires: 3 September 2026

C. Gomez
UPC
J. Crowcroft
University of Cambridge
M. Tuexen
Muenster Univ. of Appl. Sciences
March 2026

TCP ACK Rate Request Option
draft-ietf-tcpm-ack-rate-request-11

Abstract

TCP Delayed Acknowledgments (ACKs) is a widely deployed mechanism that allows reducing protocol overhead in many scenarios. However, Delayed ACKs may also contribute to suboptimal performance. When a relatively large congestion window (cwnd) can be used, less frequent ACKs may be desirable. On the other hand, in relatively small cwnd scenarios, eliciting an immediate ACK may avoid unnecessary delays that may be incurred by the Delayed ACKs mechanism. This document specifies the TCP ACK Rate Request (TARR) option. This option allows a sender to request the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver.

Table of Contents

1. Introduction

Delayed Acknowledgments (ACKs) were specified for TCP with the aim to reduce protocol overhead [RFC1122]. With Delayed ACKs, a TCP delays sending an ACK by up to 500 ms (often 200 ms, with lower values in recent implementations such as ~50 ms also reported), and typically sends an ACK for at least every second segment received in a stream of full-sized segments. This allows combining several segments into a single one (e.g. the application layer response to an application layer data message, and the corresponding ACK), and also saves up to one of every two ACKs, under many traffic patterns (e.g. bulk transfers). The "SHOULD" requirement level for implementing Delayed ACKs in RFC 1122 (subsequently reinforced in RFC 5681), along with its expected benefits, has led to a widespread deployment of this mechanism.

However, there exist scenarios where Delayed ACKs contribute to suboptimal performance. We next roughly classify such scenarios into two main categories, in terms of the congestion window (cwnd) size and the Maximum Segment Size (MSS) that would be used therein: i) "large" cwnd scenarios (i.e. cwnd >> MSS), and ii) "small" cwnd scenarios (e.g. cwnd up to ~MSS).

In "large" cwnd scenarios, increasing the number of data segments after which a receiver transmits an ACK beyond the typical one (i.e. 2 when Delayed ACKs are used) may provide significant benefits. One example is mitigating performance limitations due to asymmetric path capacity (e.g. when the reverse path is significantly limited in comparison to the forward path) [RFC3449]. Another advantage is reducing the computational cost both at the sender and the receiver, and reducing network packet load, due to the lower number of ACKs involved.

In many "small" cwnd scenarios, a sender may want to request the receiver to acknowledge a data segment immediately (i.e. without the additional delay incurred by the Delayed ACKs mechanism). In high bit rate environments (e.g. data centers), a flow’s fare share of the available Bandwidth Delay Product (BDP) may be in the order of one MSS, or even less. For an accordingly set cwnd value (e.g. cwnd up to MSS), Delayed ACKs would incur a delay that is several orders of magnitude greater than the Round Trip Time (RTT), severely degrading performance. Note that the Nagle algorithm may produce the same effect for some traffic patterns in the same type of environments [RFC8490]. In addition, when transactional data exchanges are performed over TCP, or when the cwnd size has been reduced, eliciting an immediate ACK from the receiver may avoid idle times and allow timely continuation of data transmission and/or cwnd growth, contributing to maintaining low latency.

Further "small" cwnd scenarios can be found in Internet of Things (IoT) environments. Many IoT devices exhibit significant memory constraints, such as only enough RAM for a send buffer size of 1 MSS [RFC9006]. In that case, if the data segment does not elicit an application-layer response, the Delayed ACKs mechanism unnecessarily contributes a delay equal to the Delayed ACK timer to ACK transmission. The sender cannot transmit a new data segment until the ACK corresponding to the previous data segment is received and processed.

With the aim to provide a tool for performance improvement in both "large" and "small" cwnd scenarios, this document specifies the TCP ACK Rate request (TARR) option. This option allows a sender to request the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver. Therefore, TARR allows to override the Delayed ACKs mechanism [RFC1122] (while still complying with the maximum delay to send an ACK of 500 ms). However, Standards Track TCP specifications other than RFC 1122 and some Informational specifications that recommend or mandate triggering ACKs in special conditions prevail over TARR (see section 3.2 and Appendix B).

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. TCP ACK Rate Request Functionality

A TCP endpoint SHOULD announce that it supports the TARR option by including the TARR option format (with the appropriate Length value, see Section 4) in packets that have the SYN bit set.

In some cases (e.g. when SYN cookies are used [RFC4987]), the client MAY announce that it supports the TARR option in packets subsequent to the SYN packet. (Note that announcing TARR option support on the ACK in the three-way handshake is not reliable.)

Upon reception of a segment announcing support of the TARR option, a TARR-option-capable endpoint MUST announce support of the TARR option as well by including it in the next segment to be sent.

The next two subsections define the sender and receiver behaviors for devices that support the TARR option, respectively.

3.1. Sender behavior

A TCP sender MUST NOT include the TARR option in TCP segments to be sent if the TCP receiver does not support the TARR option.

A TCP sender MAY request a TARR-option-capable receiver to modify the ACK rate of the latter to one ACK every R data segments received from the sender. This request is performed by the sender by including the TARR option in the TCP header of a segment. The TARR option carries the R value requested by the sender (see section 4).

A TCP sender SHOULD only request an ACK rate greater than 2 if the expected time between consecutively received ACKs is smaller than 1 ms or if the TCP sender uses pacing (see section 5.1).

A TCP sender MUST NOT communicate a value of R corresponding to an amount of data bytes to be acknowledged at once by the receiver greater than the last known rwin size or greater than the current cwnd size.

Note that, at a given moment, the rwin size may have changed from the last rwin size known by the sender. In such case: i) if rwin size has increased, the sender will not request an R value corresponding to an amount of data bytes to be acknowledged at once greater than the current rwin size; ii) if rwin size has decreased, a request of an R value corresponding to an amount of data bytes to be acknowledged at once greater than the current rwin size will be ignored by the receiver (see Section 3.2, second paragraph).

cwnd may decrease for a number of reasons, including congestion control [RFC5681] and rwin decrease. If cwnd decreases and the new cwnd value is smaller than the number of bytes corresponding to the last R requested, the next segment from the sender SHOULD carry a TARR option with an R value corresponding to a number of bytes smaller than or equal to the current cwnd.

When a TCP sender needs a data segment to be acknowledged immediately by a TARR-option-capable receiving TCP, without modifying the steady state ACK rate being used by the receiver, the sender includes the TARR option in the TCP header of the data segment, with a value of R equal to 0. Requesting an immediate ACK from the receiver can help reduce the time it takes to detect and/or recover from packet loss.

When a retransmission is triggered by retransmission timer expiration or by Tail Loss Probe (TLP) Probe Timeout (PTO) expiration [RFC8985], if the sender knows that the TCP receiver is TARR-capable, and the last R value requested by the sender different from zero is greater than 2, the segment carrying retransmitted data SHOULD carry a TARR

option with R set to 1. (Note: In any other circumstances, a TCP segment carrying retransmitted data is not required to include a TARR option.) Once the sender determines that all retransmitted data has been acknowledged, the first segment carrying only new data SHOULD carry a TARR option with R set to 2. When a TARR option with R=2 is sent, the receiver is requested to revert to default Delayed ACKs operation [RFC 1122].

Note that, even if TARR has been successfully used in an on-going TCP connection, the path from source to destination may change to a new one where packets carrying the TARR option might not be supported (e.g., due to a TARR-unfriendly middlebox). Not including the TARR option in a retransmitted packet, or in the first packet carrying only new data, is a conservative approach that may prevent such packets from being discarded in a possible such new path.

3.2. Receiver behavior

A receiving TCP conforming to this specification MUST process a TARR option present in a received segment.

A TARR-option-capable receiving TCP MUST ignore a TARR option requesting a value of R corresponding to an amount of data bytes to be acknowledged at once greater than its current rwin size. Otherwise, a TARR-option-capable receiving TCP SHOULD modify its ACK rate to one ACK every R received data segments from the sender. The receivers’s count of data segments received from the sender is reset every time that an ACK is sent for any reason.

If the rwin size of a TARR-option-capable TCP decreases to a value lower than the amount of data bytes to be acknowledged at once for the latest R requested, the amount of data bytes acknowledged at once by an ACK sent by the receiving TCP MUST NOT exceed its current rwin size.

If a TARR-option-capable TCP receives a segment carrying the TARR option with R=0, the receiving TCP SHOULD send an ACK immediately while keeping its steady state ACK rate.

Following the behavior specified in RFC 5681, in order to aid the sender in segment loss detection and repair, a TARR-option-capable receiver SHOULD send a duplicate ACK immediately when an out-of-order segment arrives [RFC5681], regardless of the last ACK rate requested by the sender. After sending a duplicate ACK, the receiver MAY send the next non-duplicate ACK after R data segments received. In addition, a TARR-option-capable receiver SHOULD send an immediate ACK when the incoming segment fills in all or part of a gap in the sequence space [RFC5681], regardless of the last ACK rate requested by the sender.

Following RFC 5961, in order to protect from an attack whereby an off-path attacker may inject a spoofed TCP segment hoping to cause the connection to be torn down, if a TARR-option-capable TCP receiver receives a segment with the RST bit set, and the sequence number does not exactly match the next expected sequence value, yet is within the current receive window, the receiver MUST send an ACK (called "challenge ACK" [RFC5961]), regardless of the last requested ACK rate.

A TARR-option-capable receiver in AccECN mode MUST comply with the rules specified in draft-ietf-tcpm-accurate-ecn, section 3.2.2.5.1, on when it emits an ACK, regardless of the last ACK rate requested. That is, such a receiver SHOULD emit an ACK (Change-Triggered ACK) whenever a data packet marked Congestion Experienced (CE) arrives after the previous packet was not CE, and it MUST emit an ACK (Increment-Triggered ACK) if ’n’ CE marks have arrived since the previous ACK [I-D.ietf-tcpm-accurate-ecn]. (If there is unacknowledged data at the receiver, ’n’ SHOULD be 2. If there is no unacknowledged data at the receiver, ’n’ SHOULD be 3 and MUST be no less than 3. In either case, ’n’ MUST be no greater than 7.)

When Data Center TCP (DCTCP) is used, if the receiver is TARR-option-capable, it MUST comply with the rules specified in section 3.2 of RFC 8257 that produce ACKs, regardless of the last ACK rate requested. This is in order to allow the sender to be able to handle congestion by determining the number of bytes sent that encountered congestion [RFC8257].

Note also that the receiver might be unable to send ACKs at the requested rate (e.g., due to lack of resources). On the other hand, the receiver might opt not to fulfill a request for security reasons (e.g., to avoid or mitigate an attack by which a large number of senders request disabling delayed ACKs simultaneously and send a large number of data segments to the receiver).

In any case, as specified in RFC 9293, the delay for an ACK MUST be less than 0.5 seconds.

The request to modify the ACK rate of the receiver holds until the next segment carrying a TARR option is received.

4. Option Format

The TARR option presents two different formats that can be identified by the corresponding format length. For packets that announce TARR option support by their senders, the TARR option has the format shown in Fig. 1.

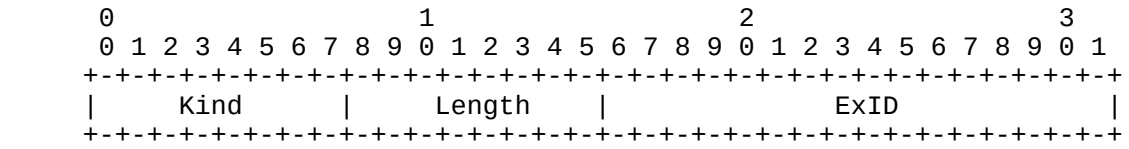

Figure 1: Format used to announce support of the TARR option by the sender.

Kind: The Kind field value is 254.

Length: The Length field value is 4 bytes.

ExID: The experiment ID field size is 2 bytes, and its value is 0x00AC.

When the sender requests an ACK rate of R, the TARR option has the format and content shown in Fig. 2.

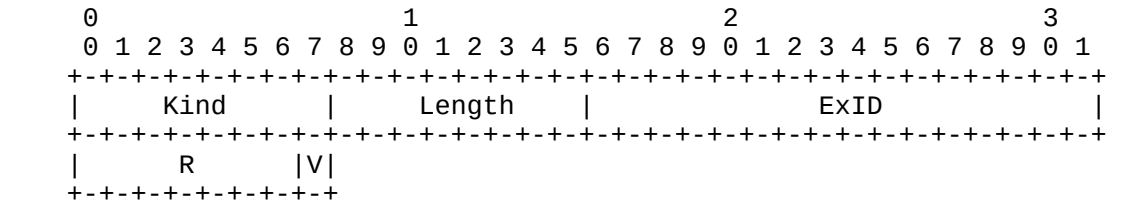

Figure 2: TCP ACK Rate Request option format.

Kind: The Kind field value is 254.

Length: The Length field value is 5 bytes.

ExID: The experiment ID field size is 2 bytes, and its value is 0x00AC.

R: The size of this field is 7 bits. The field carries the binary encoding of the ACK rate requested by the sender. The maximum value of R is 127.

V (reserVed): The size of this field is 1 bit. This field is reserved for future use.

5. Issues of Stretch ACKs

The TARR option can be used to increase the number of data segments after which a receiver sends an ACK. ACKs that acknowledge more than two previously unacknowledged data segments are known as "Stretch ACKs" [StrACKs]. Stretch ACKs have been reported to produce a number of undesirable effects [RFC2525], including increased TCP sender burst size (which may be inflicted also on other flows), increased time for TCP to open the cwnd, and reduced frequency of RTT samples. Note that, as per this specification, R values up to 127 are possible.

5.1. Sender burstiness

Increased TCP sender burstiness may contribute to router queue overflow and packet loss. One technique that a sender SHOULD use to mitigate the sender burstiness that stems from Stretch ACKs is TCP Sender Pacing [RFC2760]. This technique allows to space the transmission of data segments over a given time interval (e.g., the RTT). TCP Sender Pacing requires an algorithm to determine the appropriate data segment transmission rate, which needs to be commensurate with the R value being used.

5.2. Slow cwnd opening

During slow start, cwnd increases by up to Sender Maximum Segment Size (SMSS) upon receipt of an ACK covering new data [RFC5681]. However, Stretch ACKs (or even Delayed ACKs) reduce the amount of ACKs received by the sender, thus reducing the rate of cwnd growth, increasing transfer time and reducing throughput, when compared with sending an ACK for each incoming data segment. Note that, while Appropriate Byte Counting (ABC) [RFC3465] might be used to address this problem, it remains an experimental mechanism, not fully included in RFC 5681, which specifies standard TCP congestion control.

In order to avoid slow cwnd opening, a TCP sender SHOULD NOT use the TARR option to produce Stretch ACKs during Slow Start. While there exist TCP receiver implementations that send one ACK per received data segment during Slow Start, a TCP sender MAY use the TARR option with R=1 for data segments transmitted during Slow Start.

5.3. Lower frequency of RTT samples

When TARR produces Stretch ACKs, the number of RTT samples that the sender can obtain decreases. This reduces the responsiveness of the RTT estimate to path RTT changes. Therefore, time-based packet loss detection could either unnecessarily delay ACKs or could result in spurious retransmissions. In order to limit this issue, when there are segments in flight, a sender needs to trigger a sufficient number of ACKs per round trip. (This number depends on the specific scenario, with the best currently known value for such number being roughly in the range of at least 1-4). This can be achieved by i) sending a data segment with the TARR option with R=0 at the required rate or ii) using a greater R value intended to produce the required ACK rate.

6. Changing the ACK rate during the lifetime of a TCP connection

In some scenarios, setting the ACK rate once for the whole lifetime of a TCP connection may be suitable. However, there are also cases where it may be desirable to modify the ACK rate during the lifetime of a connection.

The ACK rate to be used may depend on the cwnd value used by the sender, which can change over the lifetime of a connection. cwnd will start at a low value and grow rapidly during the slow-start phase, then settle into a reasonably consistent range for the congestion-avoidance phase - assuming the underlying bandwidth-delay product (BDP) remains constant. Phenomena such as routing updates, link capacity changes or path load changes may modify the underlying BDP significantly. The cwnd should be expected to change accordingly, prompting the need for ACK rate updates. cwnd may also change due to relatively sporadic phenomena, such as retransmission timer expiration, regardless of the steady-state cwnd value for a given path; in such cases, ACK rate updates may be needed as well. Note that the sender may opt to request an ACK rate that it considers appropriate at any moment.

TARR can also be used to suppress Delayed ACKs in order to allow measuring the RTT of each packet in specific intervals (e.g., during flow start-up), and allow a different ACK rate afterwards.

A Linux receiver has a heuristic to detect slow start and suppress Delayed ACKs just for that period. However, some slow start variants (e.g., HyStart, HyStart++, etc.) may alter the ending of slow start, thus confusing the heuristics of the receiver [I-D.ietf-tcpm-hystartplusplus]. To avoid slow start sender behavior ossification, an explicit signal such as TARR may be useful.

In some scenarios, the sender may notice that the ACKs it receives cover more segments than the ACK rate requested. This may be due to two reasons: i) ACK filtering or ACK decimation [RFC 3449] is occurring en route; or ii) the receiver uses Large Receive Offload (LRO). If the reason is known or suspected by the sender, in the former, the sender may decide to reduce the ACK frequency to reduce receiver workload and network load up to the ACK filtering or ACK decimation point. In the latter, the sender may want to increase the ACK frequency to compensate for the impact of the LRO engine on the ACK flow.

Future TCP specifications may also permit Congestion Experienced (CE) marks to appear on pure ACKs [I-D.ietf-tcpm-generalized-ecn]. This might involve more frequent ACK rate updates (e.g., once an RTT), as the sender probes around an operating point.

7. Socket API Considerations

This section describes how the socket API can be extended to provide a way for an application to use the functionality described in this document.

This section is informational only.

The API described in this section can change in a non-backwards compatible way during the evolution of this document due to changed functionality or gained experience during the implementation.

7.1. Socket Options

Table 1 provides an overview of the IPPROTO_TCP-level socket options defined in this section.

Option Name	Data Type	Set	Get
TCP_ACK_RATE_REQ_ENABLE	int	X	X
TCP_ACK_RATE_REQ_PROCESS	int	X	
TCP_ACK_RATE_REQ_SET	uint32_t	X	

Table 1: Socket Options

7.1.1. Enable the TCP ACK Rate Request Extension (TCP_ACK_RATE_REQ_ENABLE)

When using setsockopt() with the IPPROTO_TCP-level socket option TCP_ACK_RATE_REQ_ENABLE for a TCP socket in the state CLOSED or LISTEN, the negotiation of the TARR feature can be disabled by providing a zero option_value and enabled with a non-zero option_value.

When using getsockopt() with this socket option for a connected TCP socket, the option_value of zero is returned, if TARR support has not been negotiated for the TCP connection or non-zero if TARR support has been negotiated.

7.1.2. Process Received ACK Rate Requests (TCP_ACK_RATE_REQ_PROCESS)

When using setsockopt() with the IPPROTO_TCP-level socket option TCP_ACK_RATE_REQ_PROCESS for a TCP socket, the processing of incoming TARR options is enabled by providing an non-zero option_value and disabled by providing a zero option_value.

The default value of this socket option is that incoming options are processed.

For accepted sockets, this socket option is inherited from the listening socket.

7.1.3. Request an ACK Rate (TCP_ACK_RATE_REQ_SET)

When using setsockopt() with the IPPROTO_TCP-level socket option TCP_ACK_RATE_REQ_SET for a connected TCP socket, a TARR request for the rate specified in the option_value will be sent with the next TCP segment.

8. IANA Considerations

This document specifies a new TCP option (TCP ACK Rate Request) that uses the shared experimental options format [RFC6994], with ExID in network-standard byte order.

IANA has assigned the ExID value 0x00AC for the TCP option specified in this document.

9. Security Considerations

The TARR option opens the door to new security threats. This section discusses such new threats, and suggests mitigation techniques.

An attacker might be able to impersonate a legitimate sender, and forge an apparently valid packet intended for the receiver. In such case, the attacker may mount a variety of harmful actions. By using TARR, the attacker may intentionally communicate a bad R value to the latter with the aim to damage communication or device performance. For example, in a small cwnd scenario, using a too high R value may lead to exacerbated RTT increase and throughput decrease. In other scenarios, a too low R value may contribute to depleting the energy of a battery-operated receiver at a faster rate or may lead to increased network packet load.

While Transport Layer Security (TLS) [RFC8446] is strongly recommended for securing TCP-based communication, TLS does not protect TCP headers, and thus cannot protect the TARR option fields carried by a segment. One approach to address the problem is using network-layer protection, such as Internet Protocol Security (IPsec) [RFC4301]. Another solution is using the TCP Authentication Option (TCP-AO), which provides TCP segment integrity and protection against replay attacks [RFC5925].

While it is relatively hard for an off-path attacker to attack an unprotected TCP session, it is RECOMMENDED for a TARR receiver to use the guidance and attack mitigation given in [RFC5961]. The TARR option MUST be ignored on a packet that is deemed invalid.

A TARR receiver might opt not to fulfill a request to avoid or mitigate an attack by which a large number of senders request disabling delayed ACKs simultaneously and send a large number of data segments to the receiver (see Section 3.2).

10. Acknowledgments

Bob Briscoe, Jonathan Morton, Richard Scheffenegger, Neal Cardwell, Michael Tuexen, Yuchung Cheng, Matt Mathis, Jana Iyengar, Gorry Fairhurst, Stuart Cheshire, Yoshifumi Nishida, Michael Scharf, Ian Swett, and Martin Duke provided useful comments and input for this document. Jonathan Morton and Bob Briscoe provided the main input for Section 6. Section 5.3 has been inspired by related guidance (for QUIC) included in draft-ietf-quic-ack-frequency, and discussion in the IETF QUIC working group.

Carles Gomez has been funded in part by the Spanish Government MCIU/AEI/10.13039/501100011033/FEDER/UE through projects PID2019-106808RA-I00, PID2023-146378NB-I00, and by Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya through grant number 2017 SGR 376 and grant number 2021 SGR 00330.

11. References

11.1. Normative References

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2760] Allman, M., Ed., Dawkins, S., Glover, D., Griner, J., Tran, D., Henderson, T., Heidemann, J., Touch, J., Kruse, H., Ostermann, S., Scott, K., and J. Semke, "Ongoing TCP Research Related to Satellites", RFC 2760, DOI 10.17487/RFC2760, February 2000, <https://www.rfc-editor.org/info/rfc2760>.

[RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <https://www.rfc-editor.org/info/rfc3465>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <https://www.rfc-editor.org/info/rfc5961>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <https://www.rfc-editor.org/info/rfc6994>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <https://www.rfc-editor.org/info/rfc8257>.

[RFC8985] Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <https://www.rfc-editor.org/info/rfc8985>.

11.2. Informative References

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., K端hlewind, M., and R. Scheffenegger, "More Accurate Explicit Congestion Notification (AccECN) Feedback in TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-accurate-ecn-34, 10 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn-34>.

[I-D.ietf-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", Work in Progress, Internet-Draft, draft-ietf-tcpm-generalized-ecn-17, 21 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-generalized-ecn-17>.

[I-D.ietf-tcpm-hystartplusplus]
Balasubramanian, P., Huang, Y., and M. Olson, "HyStart++: Modified Slow Start for TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-hystartplusplus-14, 27 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-hystartplusplus-14>.

[RFC2525] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP Implementation Problems", RFC 2525, DOI 10.17487/RFC2525, March 1999, <https://www.rfc-editor.org/info/rfc2525>.

[RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC4782] Floyd, S., Allman, M., Jain, A., and P. Sarolahti, "Quick-Start for TCP and IP", RFC 4782, DOI 10.17487/RFC4782, January 2007, <https://www.rfc-editor.org/info/rfc4782>.

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <https://www.rfc-editor.org/info/rfc5690>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <https://www.rfc-editor.org/info/rfc8490>.

[RFC9006] Gomez, C., Crowcroft, J., and M. Scharf, "TCP Usage Guidance in the Internet of Things (IoT)", RFC 9006, DOI 10.17487/RFC9006, March 2021, <https://www.rfc-editor.org/info/rfc9006>.

[StrACKs] Paxson, V., "Automated packet trace analysis of TCP implementations", 1997.

Appendix A. Relation between the present document and RFC 5690

A previously published document, entitled "Adding Acknowledgment Congestion Control to TCP" [RFC5690], includes functionality similar to some aspects of the present document. However, the motivation, main goals, and use cases of both documents are almost orthogonal. In fact, some features of the present document were not considered in [RFC5690]. This section compares the main features of RFC 5690 and the present document.

A.1. Motivation, goals and features

RFC 5690 is an informational document that describes a possible congestion control mechanism for TCP ACKs. The main goal is to reduce ACK traffic when there is congestion on the reverse path in order to reduce the congestion. The mechanism includes: i) a component for the TCP sender to detect lost and ECN-marked pure ACKs, ii) a mechanism for adjusting the ACK Ratio, iii) a method to discover the support of the ACK congestion control mechanism by an endpoint (by means of a new TCP option), and iv) a method for the TCP sender to inform the TCP receiver of a new value for the ACK Ratio (by means of a second new TCP option). As of the writing, and to the best knowledge of the authors, RFC 5690 has not been implemented. Option Kind values for the new TCP options described in RFC 5690 have neither been allocated by IANA.

The present document defines the TARR option. While it can be used to reduce network load, its primary focus is rather on end-to-end performance and end-system resource conservation. TARR serves two purposes: i) allowing a sender to request a given ACK ratio from the receiver, and ii) allowing a sender to request an immediate ACK, without modifying the steady state ACK ratio. The latter is not supported by RFC 5690. On the other hand, TARR might be used as a component of other mechanisms (e.g. an ACK congestion control mechanism). However, such mechanisms are out of the scope of the present document.

A.2. New TCP option details

As part of the ACK congestion control mechanism, RFC 5690 proposes the use of two new TCP options: one intented to announce support of TCP ACK Congestion Control, and another one which is used by the sender to communicate the ACK ratio to the receiver. The former can only be sent on packets that have the SYN bit set. In the latter, a one-byte field is used to carry the ACK ratio, but the encoding to be used for this field is not defined.

The present document uses a single TCP experimental option Kind value (following RFC 6994) for both announcing support of the TARR option, and for communicating the requested ACK ratio. In the present document, announcing support of the TARR option may be done in packets that do not have the SYN bit set, with the aim to alleviate the need for TCP option space in SYN packets. In contrast with RFC 5690, the encoding to be used for the ACK ratio field is specified (see Section 4).

Appendix B. Other documents that provide rules on sending ACKs

B.1. Standards Track documents

RFC 1122, Section 4.2.2.21 allows (via a normative "MAY") a receiver to send an ACK in response to an out-of-order data segment, i.e., a segment that is in the window, but not at the left window edge (MAY-13 in RFC 9293). RFC 2018 states that, in such circumstance, the receiver "SHOULD send an ACK", including a SACK option, "for every valid segment that arrives containing new data".

Following similar principles, RFC 5681 states that, in order to aid the sender in segment loss detection and repair, a receiver SHOULD send a duplicate ACK immediately when an out-of-order segment arrives. In addition, a receiver SHOULD send an immediate ACK when the incoming segment fills in all or part of a gap in the sequence space.

RFC 5961 updates RFC 793 to state that a TCP that receives a segment with the RST bit "SHOULD implement" a set of instructions which include: "If the RST bit is set and the sequence number does not exactly match the next expected sequence value, yet is within the current receive window, TCP MUST send an acknowledgment (challenge ACK)".

draft-ietf-tcpm-accurate-ecn, section 3.2.2.5.1, provides "The following rules define when the receiver of a packet in AccECN mode emits an ACK:

Change-Triggered ACKs: An AccECN Data Receiver SHOULD emit an ACK whenever a data packet marked CE arrives after the previous packet was not CE.

Even though this rule is stated as a "SHOULD", it is important for a transition to trigger an ACK if at all possible, The only valid exception to this rule is given below these bullets.

For the avoidance of doubt, this rule is deliberately worded to apply solely when _data_ packets arrive, but the comparison with the previous packet includes any packet, not just data packets.

Increment-Triggered ACKs: An AccECN receiver of a packet MUST emit an ACK if ’n’ CE marks have arrived since the previous ACK. If there is unacknowledged data at the receiver, ’n’ SHOULD be 2. If there is no unacknowledged data at the receiver, ’n’ SHOULD be 3 and MUST be no less than 3. In either case, ’n’ MUST be no greater than 7."

B.2. Informational documents

RFC 5690 describes a possible congestion control mechanism for TCP ACKs (see Appendix A).

RFC 8257 describes Data Center TCP (DCTCP). In order to handle congestion, this scheme requires that the sender must be able to determine the number of bytes sent that encountered congestion. To this end, "DCTCP introduces a new Boolean TCP state variable, DCTCP Congestion Encountered (DCTCP.CE)". And follows: "When receiving packets, the CE codepoint MUST be processed as follows:

1. If the CE codepoint is set and DCTCP.CE is false, set DCTCP.CE to true and send an immediate ACK.

2. If the CE codepoint is not set and DCTCP.CE is true, set DCTCP.CE to false and send an immediate ACK.

3. Otherwise, ignore the CE codepoint.

Since the immediate ACK reflects the new DCTCP.CE state, it may acknowledge any previously unacknowledged packets in the old state. This can lead to an incorrect rate computation at the sender per Section 3.3. To avoid this, an implementation MAY choose to send two ACKs: one for previously unacknowledged packets and another acknowledging the most recently received packet."

B.3. Experimental documents

RFC 4782 describes the experimental Quick-Start mechanism for transport protocols, and specifies its use with TCP. Quick-Start may produce a sudden increase of pure ACKs transmitted on the reverse path. The document proposes that "In the absence of congestion control for acknowledgement traffic, the TCP receiver could limit its sending rate for ACK packets sent in response to Quick-Start data packets". The document proposes that the receiver can acknowledge the first Quick-Start data packet, and every succeeding K data packets, and gives a formula to determine K.

Note that TARR can be used to allow a Quick-Start sender to request the ACK rate to be used by the receiver.

Appendix C. Impact of TARR in the presence of elements that modify the ACK rate

ACK filtering and ACK decimation are techniques that may be used by the receiver or by a middlebox intended to reduce the number of TCP ACKs [RFC 3449].

In ACK filtering, the transmit queue of the device performing ACK filtering is inspected. When there are several ACKs stored in that queue for a given connection, some or all of the older ACKs of the connection are deleted. In contrast, ACK decimation drops ACKs from a queue without a proper control on which ACKs are being dropped.

Using TARR (with R > 2) in the presence of elements performing ACK filtering may contribute to producing Stretch ACKs (see Section 5), but there will be at least one ACK sent per sender cwnd. However, ACK decimation may drop all ACKs corresponding to a cwnd of data, thus producing retransmission timer expiration. Note that, in this case, absence of TARR would elicit a greater number of ACKs from the receiver, leading to a greater probability that at least one of the ACKs for a given cwnd of data would not be dropped. This motivates the sender behavior in Section 3.1 to request the receiver to revert to default Delayed ACKs operation [RFC 1122] after retransmission timer expiration when using TARR (with R > 2).

Receiver-side aggregation techniques such as LRO may also reduce the number of ACKs. Using TARR with such a receiver may also reduce further the number of ACKs. In some cases, this might also lead to not eliciting at least one ACK per cwnd of data. The same measure of reverting to default Delayed ACKs operation will prevent undesired future consequences of using TARR in the same TCP connection.

Authors' Addresses

Carles Gomez
UPC
C/Esteve Terradas, 7
08860 Castelldefels
Spain
Email: carles.gomez@upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge
United Kingdom
Email: jon.crowcroft@cl.cam.ac.uk

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany
Email: tuexen@fh-muenster.de