TCPM Working Group
Internet-Draft
Intended status: Experimental
Expires: May 7, 2020

C. Gomez
UPC
J. Crowcroft
University of Cambridge

November 4, 2019

TCP ACK Pull
draft-gomez-tcpm-ack-pull-01

Abstract

Delayed Acknowledgments (ACKs) allow reducing protocol overhead in many scenarios. However, in some cases, Delayed ACKs may significantly degrade network and device performance in terms of link utilization, latency, memory usage and/or energy consumption. This document defines the TCP ACK Pull (AKP) mechanism, which allows a sender to request the ACK for a data segment to be sent without additional delay by the receiver. AKP makes use of one of the reserved bits in the TCP header, which is defined in this specification as the AKP flag.

Table of Contents

1. Introduction

Delayed Acknowledgments (ACKs) were specified 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 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 it 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, along with its expected benefits, has led to a widespread deployment of this mechanism.

However, there exist traffic patterns and scenarios for which Delayed ACKs can actually be detrimental to performance. When a segment carrying a message of a size up to one Maximum Segment Size (MSS) is transferred, if the message does not elicit an application-layer response, and a second data segment is not transferred earlier than the Delayed ACK timeout, the ACK is unnecessarily delayed, with a number of negative consequences.

When the Nagle algorithm is used, in some cases the sender may be prevented from sending more data while awaiting the Delayed ACK. In some high bit rate environment (e.g. Gigabit Ethernet) use cases, such a delay may be very large, and link utilitzation may be

dramatically reduced, as the Delayed ACK timeout is several orders of magnitude greater than the Round Trip Time (RTT) [RFC8490].

Delayed ACKs are also detrimental in Internet of Things (IoT) scenarios, where TCP is being increasingly used [I-D.ietf-lwig-tcp-constrained-node-networks]. Many IoT devices, such as sensors, transfer small messages (e.g. containing sensor readings) rather infrequently, therefore if the receiver uses Delayed ACKs, the ACK will often be unnecessarily delayed. The sender cannot release the memory resources associated to a transferred data segment until the ACK is received and processed. This may be a problem for many IoT devices, which are typically memory-constrained, and may even lead to subsequent packet drops if their scarce memory resources are blocked while awaiting an ACK. Moreover, if the IoT device uses a radio interface for communication, in some scenarios Delayed ACKs will lead to increased energy consumption (e.g. with the radio interface of the device staying in receive mode while awaiting the ACK). Since many IoT devices run on small batteries, the device lifetime may be significantly decreased. Furthermore, the delay suffered by the ACK may interact negatively with layer two mechanisms, especially in wireless network technologies where devices remain in low-power states for long intervals [RFC 8352], potentially leading to a further exacerbated delay (by even one or more orders of magnitude).

One approach that cannot be recommended as a general solution to solve the described problems is disabling Delayed ACKs at the receiving TCP. In fact, the latter may interact with a wide variety of devices and many of those may still benefit from the advantages of Delayed ACKs. In addition, in some cases, a sender may offer a mixed traffic pattern comprising single data segments that will lead to unnecessarily delayed ACKs, with other data segments upon which Delayed ACKs will act as intended. Therefore, the solution has to be provided at a per-segment granularity.

Since the presented problem is about low performance in various scenarios, another requirement for the solution is to provide a sender with a mechanism to request an immediate ACK for a data segment without incurring overhead in terms of header size increase or additional packets sent. For example, in IoT scenarios, every additional communicated byte consumes scarce resources (e.g. energy, bandwidth, computational resources); even further, each additional communicated packet may involve significant energy overhead.

This document defines the TCP ACK Pull (AKP) mechanism and an AKP flag in the TCP header. AKP allows a sender to request an ACK to be sent by a receiving TCP without additional delay upon reception of a data segment, by setting the AKP flag in that data segment. The AKP

flag uses one of the reserved bits in the TCP header. More specifically, the AKP flag uses bit 6 of byte 13 of the TCP header.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL","SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. ACK Pull Mechanism

When a TCP sender needs a data segment to be acknowledged by the receiving TCP without additional delay, the sender sets the AKP flag of the data segment TCP header. A receiving TCP conforming to this specification MUST process the AKP flag of a received segment. If the AKP flag is set, the receiving TCP MUST send an ACK without additional delay, regardless of whether the receiving TCP uses the Delayed ACKs mechanism.

4. The ACK Pull Flag

The AKP flag is defined as bit number 6 of the 13th byte of the TCP header. Figure 1 illustrates bytes 13 and 14 of the TCP header, including the AKP flag.

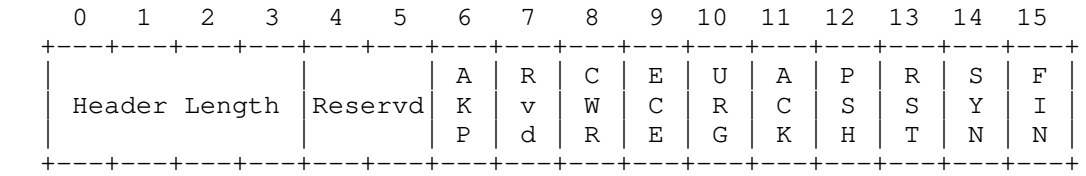

Figure 1: Definition of the AKP field within bytes 13 and 14 of the TCP Header.

(Note: as of the writing, bit 7 in the above figure is reserved, although this may change with the publication of [I-D.ietf-tcpm-accurate-ecn].)

5. IANA Actions

This document assigns bit 6 of the TCP header flags to the AKP flag. This flag will be defined as shown in Figure 2:

Bit	Name	Reference
6	AKP (ACK Pull)	RFC XXXX

Figure 2

[TO BE REMOVED: IANA is requested to update the existing entry in the Transmission Control Protocol (TCP) Header Flags registration (https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml#tcp-header-flags-1) for Bit 6 to 'AKP (ACK Pull)'.]

6. Security Considerations

TCP ACK Pull introduces a possible Denial of Service (DoS) attack on a resource-constrained receiver. An attacker might send a large number of messages to a victim node, requesting an immediate ACK in response to each one of them. This attack is easily avoided by ignoring the TCP ACK Pull flag.

7. Acknowledgments

Stuart Cheshire, Ted Lemon, Michael Scharf, and Christoph Paasch participated in a discussion that was seminal to this document.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Ciencia, Innovacion y Universidades) through the Jose Castillejo grant CAS18/00170 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE. His contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

8. Annex: Alternative approaches

Several mechanisms that have been proposed in the past allow increasing the amount of ACKs sent by the receiver. Some examples are Acknowledgment Congestion Control (AckCC) [RFC5690] and Tail Loss Probe (TLP) [I-D.ietf-tcpm-rack].

In AckCC, the sender tells the receiver the ACK Ratio R to use, where the receiver sends one ACK per R data packets received. AckCC defines a 2-byte "TCP ACK Congestion Control Permitted Option" for negotiating use of AckCC, whereas it defines a 3-byte "ACK Ratio TCP option" to communicate the ACK Ratio value from the sender to the receiver.

TLP is intended to avoid RTO-expiration-based retransmission when tail loss occurs by inducing additional ACKs at the receiver. This is achieved by sending a probe segment after a probe time-out (PTO) when data have been sent but not confirmed.

Another approach that allows eliciting an immediate ACK after sending a data segment is sending a subsequent segment carrying e.g. an already sent data byte. Another workaround, which is used in the Contiki operating system (a popular operating system for constrained devices in IoT scenarios) is to split the data to be sent into two segments of smaller size. A standard compliant TCP receiver will acknowledge the second MSS of data, which can improve throughput. However, this 'split hack' may not always work since a TCP receiver is required to acknowledge every second full-sized segment, but not two consecutive small segments. Furthermore, the overhead of sending two IP packets instead of one is another downside of the 'split hack'.

Note that all the approaches in this Annex involve increasing TCP header size of some segments, or involve sending additional packets. The main advantage of the AKP mechanism defined in this specification is allowing a sender to request an immediate ACK while incurring no overhead.

9. References

9.1. 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>.

9.2. Informative References

[I-D.ietf-lwig-tcp-constrained-node-networks] Gomez, C., Crowcroft, J., and M. Scharf, "TCP Usage Guidance in the Internet of Things (IoT)", draft-ietf-lwig-tcp-constrained-node-networks-08 (work in progress), June 2019.

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-09 (work in progress), July 2019.

[I-D.ietf-tcpm-rack]
Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "RACK: a time-based fast loss detection algorithm for TCP", draft-ietf-tcpm-rack-06 (work in progress), November 2019.

[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>.

[RFC8352] Gomez, C., Kovatsch, M., Tian, H., and Z. Cao, Ed., "Energy-Efficient Features of Internet of Things Protocols", RFC 8352, DOI 10.17487/RFC8352, April 2018, <https://www.rfc-editor.org/info/rfc8352>.

[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>.

Authors' Addresses

Carles Gomez
UPC
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge, CB3 0FD
United Kingdom

Email: jon.crowcroft@cl.cam.ac.uk