CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: February 1, 2013

C. Bormann
K. Hartke
Universitaet Bremen TZI
July 31, 2012

Congestion Control Principles for CoAP
draft-bormann-core-congestion-control-02

Abstract

The CoAP protocol needs to be implemented in such a way that it does not cause persistent congestion on the network it uses. Congestion control is a complex issue -- the proper rationale for the congestion control mechanisms chosen in CoAP is probably more material than the CoAP protocol specification itself. This informational document attempts to pull out the background material and more extensive considerations behind the CoAP congestion control mechanisms, while leaving the basic MUSTs and MUST NOTs in the main spec.

Table of Contents

1. Introduction 3
1.1. Terminology 3
1.2. Objectives 4
1.2.1. TCP-Friendliness 4
1.2.2. Actually working well 4
1.2.3. Getting actual implementation 5
2. Input 6
2.1. RFC 2914 6
2.2. RFC 5405 6
2.3. draft-eggert-core-congestion-control 7
3. coap-11 Congestion Control Principles 8
4. How do other protocols do it 11
4.1. DNS 11
4.2. SIP 11
4.3. TCP 11
4.4. HTTP 12
5. Advanced CoAP Congestion Control 14
5.1. RTT Measurement 14
5.2. Block Slow-Start 14
6. Changes Planned for Base Specifications 16
7. IANA Considerations 17
8. Security Considerations 18
9. Acknowledgements 19
10. References 20
10.1. Normative References 20
10.2. Informative References 20
Authors' Addresses 22

1. Introduction

With few exceptions, it is simply incompetent to build an implementation of a packet-based protocol without considering congestion control. Unfortunately, detailed, evidence-based knowledge about congestion control is limited to a small group of people. It has become customary for these to try to encode their knowledge into the protocol definitions, in an attempt to replace competence by conformance.

This has worked relatively well for TCP, not the least because the art of TCP implementation is itself limited to a rather small group of experts, which over the years often have acquired some knowledge of congestion control principles, complementing the desire for conformance by substantial competence again. Conversely, application developers are a much larger, much more diverse group. Worse, protocol complexity for which the rationale is not apparent to the developers might simply not be implemented. Giving congestion-unaware developers UDP sockets that are not protected by TCP's congestion control may lead to disasters.

With this background, an application protocol that is threatening to be widely deployed and does not rely on the built-in congestion control properties of TCP presents a serious worry.

This document attempts to present a more extensive rationale for CoAP's minimal, but effective congestion control design, as well as some updates to it. This rationale is not included in [I-D.ietf-core-coap] or [I-D.ietf-core-observe] as the specification is threatening to become too long with all the rationale and implementation considerations discussion already included. While the present document discusses normative statements, it is not intended to supplement or replace the normative statements in [I-D.ietf-core-coap] and [I-D.ietf-core-observe], but just to provide additional explanation.

((Editorial note: the updates to the mandates discussed here partially still need to make it into the next version of [I-D.ietf-core-coap]. A summary of the updates needed is in Section 6.))

1.1. Terminology

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] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

(Note that this document is itself informational, but it is discussing normative statements.)

The term "byte" is used in its now customary sense as a synonym for "octet".

1.2. Objectives

The objectives of adding congestion control to the CoAP protocol specification can be on two different levels, with one additional (third) consideration.

1.2.1. TCP-Friendliness

Much of the knowledge that the IETF has accumulated on congestion control focuses on not being worse than its flagship transport protocol, TCP, and being "fair" to instances of TCP competing for capacity. Since fairness is not really a well-defined term, we reduce it to "friendliness".

One objective of this document is to discuss how CoAP can be employed in a TCP-friendly way, and what are the minimum mandates the protocol needs to make in order to ensure this for reasonable applications.

(Note that TCP itself is not TCP-friendly when abused, e.g., when opening 10000 connections in close succession; so there will be no attempt to stay TCP-friendly when CoAP is abused, either.)

Conclusion: CoAP needs to be TCP-friendly, but probably not more so than TCP itself.

1.2.2. Actually working well

Making sure that the network continues to work well in the presence of a strong deployment of active CoAP endpoints is a much harder objective to achieve. There is only limited knowledge about the characteristics of the constrained node/networks CoAP will be used in. They might exhibit congestion in surprising ways.

It may turn out the collected wisdom that has been derived from TCP deployment experience in the mostly browser-oriented Internet does not transfer to the Internet of Things, and that we need to invent new mechanisms for the latter.

But this is research.

Imposing the need for a completed solution that meets requirements entirely unknown at this time would be an instance of the Fallacy of

Perfection [GF].

We will need to accumulate additional knowledge, on a research basis, and with experience coming in from larger CoAP deployments. One likely outcome is that constrained node/networks will simply continue to evolve to be able to cope with TCP and CoAP.

Conclusion: For now, we will focus on staying safe where TCP would have stayed safe.

1.2.3. Getting actual implementation

The protocol specification may specify whatever it wants; if there is significant complexity in implementing a mandate and the rationale is not apparent for implementers, compliance will be but a lucky coincidence - even more so in implementations for highly constrained systems. A design that achieves stable operation outside pathological situations and is implemented is preferable to a picture-perfect design that is a beautiful part of the specification and then ignored.

Binding the inevitable complexity of a congestion control scheme to mechanisms that already have to be implemented for other functional reasons seems the most fruitful approach for obtaining compliance. This consideration, together with the main design objective of CoAP - being implementable on constrained nodes and networks - has been the overriding design objective.

2. Input

The word "congestion" occurs more than a hundred times in 1id-abstracts.txt, indicating that there is a lot of documents under construction that might become relevant to this document. We select a few existing documents here and pick up a few salient points.

2.1. RFC 2914

[RFC2914], "Congestion Control Principles", is the BCP that lays out the basic principles for congestion control in the Internet. While it does allude to non-TCP protocols, it mainly focuses on TCP and TCP-like behavior.

2.2. RFC 5405

[RFC5405], "Unicast UDP Usage Guidelines for Application Designers", makes additional points for the usage of UDP. It is also a BCP document. Its considerations have mostly been made without looking at specific application protocols, and with a view to guiding application protocol developers towards congestion-controlled transport protocols (which is unfortunately not an appropriate choice for CoAP). It does consider the case of low data-volume applications (section 3.1.2 is therefore the most relevant section for this document). It clearly needs to be interpreted intelligently in order to arrive at congestion control guidelines for a new application protocol. E.g., it recommends:

> Applications that at any time exchange only a small number of UDP datagrams with a destination SHOULD still control their transmission behavior by not sending on average more than one UDP datagram per round-trip time (RTT) to a destination.

Instead, a CoAP client that does receive a response without the need for a retransmission should be able to send an ensuing request right away, without the need to do any such rate control -- this keeps the spirit, but not the letter of that requirement.

While [RFC5405] does provide a good set of "don't forget" points, some of its requirements appear to attempt to err on the side of caution, without regards to the specific characteristics of an application. Fortunately, these requirements are often phrased as a SHOULD, so it is possible to explain when and why they should not be heeded.

2.3. draft-eggert-core-congestion-control

[I-D.eggert-core-congestion-control], "Congestion Control for the Constrained Application Protocol (CoAP)", was the original document that led to CoAP's congestion control design. This document provides good historical context and should be read in conjunction with the present document. However, the "credit-based" mechanism proposed in its section 3.2 is probably too complicated to be implemented in constrained nodes; CoAP now uses a simpler algorithm that uses the information the implementation already has to keep (i.e., it is based on limiting the outstanding exchanges).

3. coap-11 Congestion Control Principles

CoAP is a protocol that attempts to minimize the complexity of its implementation. It is mainly intended for interactions that are not really flow-shaped, so traditional congestion control mechanisms simply do not have useful information to work on.

Basic CoAP [I-D.ietf-core-coap] uses a strict lock-step protocol for its requests and responses (both on the reliability layer with CON/ACK and one level higher with requests and responses), with exponential back-off in case of non-delivery. The initial timeout is dithered between 2 and 3 seconds and grows up to between 32 and 48 seconds.

This is inherently TCP-friendly, similar to the way protocols like DNS operate.

[I-D.ietf-core-coap] goes on to require:

> In order not to cause congestion, Clients (including proxies) SHOULD strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies). An outstanding interaction is either a CON for which an ACK has not yet been received but is still expected (message layer) or a request for which a response has not yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). A good value for this limit is the number 1. (Note that [RFC2616], in trying to achieve a similar objective, did specify a specific number of simultaneous connections as a ceiling. While revising [RFC2616], this was found to be impractical for many applications [I-D.ietf-httpbis-p1-messaging]. For the same considerations, this specification does not mandate a particular maximum number of outstanding interactions, but instead encourages clients to be conservative when initiating interactions.)

The rationale for this design is that it is very easy to implement for a constrained device: a constrained device will already have a hard limit on the number of slots available for initiating transactions. Similarly, even back-end systems already need to bind state to outstanding transactions; adding some form of congestion control state to these does not require maintaining new objects, just new fields. In any case, having some form of limit is not elective: in the text the SHOULD needs to be changed into a MUST, even though it may not be easy to pinpoint the exact criterion for compliance.

In the following, we refer to the initiator parameter that limits the number of outstanding interactions as NSTART.

Clients SHOULD also heed this [RFC5405] guideline:

> an application SHOULD perform congestion control over all UDP traffic it sends to a destination, independently from how it generates this traffic. For example, an application that forks multiple worker processes or otherwise uses multiple sockets to generate UDP datagrams SHOULD perform congestion control over the aggregate traffic.

Note that [RFC5405] is not explicit here with respect to what it considers to be a "destination"; it also uses the term "destination host" when it appears to provide specific discussion about all protocol entities at an IP address. [RFC5405] duly notes the failure of the congestion manager approach [RFC3124], but appears to wish it back into existence. For the purposes of CoAP, probably "destination" here should be used as with the CoAP term destination endpoint (i.e., including the UDP port number). Still, an implementation that e.g. uses a new source port per request (i.e. a new source endpoint, which is a valid strategy) probably needs to heed this SHOULD for the entirety of the combination of its own endpoint abstractions.

For certain exchanges in CoAP, there is a chance that a request would never elicit a response (e.g., due to a crashed server) but there is also no (protocol) timeout governing this exchange. Therefore, the count of outstanding interactions needs to decay at some rate; a decay rate below that at which TCP sends to a very lossy channel (e.g., 7 B/s) should be safe.

There are also some special congestion control considerations with responses to multicast requests, see [I-D.ietf-core-coap] section 4.5; servers are expected to provide estimates for group size and a target rate as well as a response size. Where those estimates are hard to come up with, a default response dithering window of 10 seconds should be added to [I-D.ietf-core-coap], as well an admonition for a client not to use multicast requests when such a default window would be way off. Finally, a server that receives another multicast request within the dithering window for a request that it already is answering SHOULD move the dithering window for its next response to after the first dithering window.

Finally, the text in [I-D.ietf-core-coap] needs to be reviewed whether it always clearly separates the discussion for avoiding network congestion from any mechanisms for avoiding server overloading.

[I-D.ietf-core-observe] adds one additional behavior: servers may send NON messages as notifications for state changes, which is

outside of exchanges that would be governed by NSTART. This functionality needs to be supported with some discussion of congestion control. Generally, servers SHOULD NOT send more than one NON message every 3 seconds on average ([RFC5405] section 3.1.2), and they SHOULD NOT send NON messages while waiting for CON messages to be acknowledged (however, CON retransmissions should send the new resource state if it has changed since, see [I-D.ietf-core-observe] section 4.5). There already was a decision to add a requirement to require sending a CON message at least every 24 hours before continuing with NON messages; probably the parameter of no more than a NON per 3 seconds should be increased for servers that check the client that rarely (e.g., to the rate at which TCP sends into a very lossy channel, e.g., 7 B/s).

4. How do other protocols do it

While CoAP congestion control could be designed from first principles, it is maybe more realistic to have a look at how other protocols address its respective version of the problem.

4.1. DNS

The DNS protocol, which in many characteristics is quite close to CoAP, does not have any explicit mechanisms for congestion control at all. Many documents consider DNS to be "sporadic messages", not worth of congestion control.

[RFC4336] says:

> (The short flows generated by request-response applications, such as DNS and SNMP, don't cause congestion in practice, and any congestion control mechanism would take effect between flows, not within a single end- to-end transfer of information.)

(This simple packet-for-packet request-response nature is now changing a bit with DNS being used for voluminous keying information and growing TXT records.)

4.2. SIP

SIP uses a 0.5 s initial timeout (T1 "RTT Estimate"), and uses binary exponential increase after that. That is similar to CoAP, but starts from a smaller initial estimate. CoAP is more conservative (initial RESPONSE_TIMEOUT is 2 s to 3 s) as we expect latencies in constrained networks to be higher than in the networks used for telephony.

4.3. TCP

A well-known problem with relying on TCP's built-in congestion control is that, even with all congestion-control mechanisms in place, simply multiplying the number of instances may lead to eventual congestion.

About a decade ago, TCP has increased its initial congestion window (IW) to about 3 full-size packets, or up to 4 packets with MSS <= 1095 bytes (which is comparable to CoAP's maximum packet sizes) [RFC3390]. As an Experimental specification, moving to an IW of 10 packets (IW10) is being examined [I-D.ietf-tcpm-initcwnd]. A related change is also planned in that document that will avoid resetting this initial window when the SYN or SYN/ACK is lost. This would mean that it is considered appropriate to send about 15 kB of data on a single connection without any congestion control feedback whatsoever,

except that some SYN+SYN/ACK exchange made it through. While [I-D.ietf-tcpm-initcwnd] is not yet approved, it is a WG document and there is widespread feeling of its inevitability even beyond the experimental status that is being planned now.

The numbers 3, 4, and 10 clearly provide some additional context for the selection of appropriate values of NSTART.

Conclusion: For now, it is probably appropriate to RECOMMEND keeping NSTART at or below a value chosen from the space between 3 and 10.

4.4. HTTP

HTTP is running on top of TCP, so it is TCP-friendly by definition. However, as HTTP 1.0 was using one TCP connection per request, and it became clear that browser usage would entail fetching many objects in parallel, congestion was still observed, and client implementations started to limit the number of simultaneously active connections to one server. Even when persistent connections were added (and later codified in HTTP 1.1) this remained a concern. Under 8.1.4 "Practical considerations", [RFC2616] defines a limit on the number of simultaneous connections from one client to one server.

> Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. A proxy SHOULD use up to 2*N connections to another server or proxy, where N is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion.

Intended as a guideline, this has been implemented to the letter in browser clients for a decade. However, using this as a hard limit is simply not appropriate for all environments. This led server implementers to widely deploy workarounds, such as splitting up a website between multiple servers ("domain sharding") in order to increase the connection concurrency.

From this historical evidence we can learn that well-meaning limitations can cause a lot of pain when implemented slavishly. The httpbis effort has learned this lesson and removed the suggestion for a hard limit (see [HTTPBISt131], [HTTPBISc715]). Note that it now says:

> Clients (including proxies) SHOULD limit the number of simultaneous connections that they maintain to a given server (including proxies).

Previous revisions of HTTP gave a specific number of connections as a ceiling, but this was found to be impractical for many applications. As a result, this specification does not mandate a particular maximum number of connections, but instead encourages clients to be conservative when opening multiple connections.

In particular, while using multiple connections avoids the "head-of- line blocking" problem (whereby a request that takes significant server-side processing and/or has a large payload can block subsequent requests on the same connection), each connection used consumes server resources (sometimes significantly), and furthermore using multiple connections can cause undesirable side effects in congested networks.

Note that servers might reject traffic that they deem abusive, including an excessive number of connections from a client.

Conclusion: There is no doubt that CoAP should follow this hard-learned expertise.

5. Advanced CoAP Congestion Control

5.1. RTT Measurement

For an initiator that plans to make multiple requests to one destination end-point, it may be worthwhile to make RTT measurements in order to obtain a better RTT estimation than that implied by the default initial timeout of 2 to 3 s. The usual algorithms for RTT estimation can be used [RFC6298], with appropriately extended default/base values. Note that such a mechanism MUST, during idle periods, decay RTT estimates that are shorter than the basic RTT estimate back to the basic RTT estimate, until fresh measurements become available again.

One important consideration not relevant for TCP is the fact that a CoAP round-trip may include application processing time, which may be hard to predict, and may differ between different resources available at the same endpoint. Servers will only trigger early ACKs (with a non-piggybacked response to be sent later) based on the default timers, e.g. after 1 s. A client that has arrived at a RTT estimate much shorter than the 2 to 3 s used as a default SHOULD therefore not expend all of its retransmissions in the shorter estimated timescale.

It may also be worthwhile to do RTT estimates not just based on information measured from a single destination endpoint, but also based on entire hosts (IP addresses) and/or complete prefixes (e.g., maintain an RTT estimate for a whole /64). The exact way this can be used to reduce the amount of state in an initiator is for further study.

5.2. Block Slow-Start

The CoAP protocol is not optimized for making good use of available network capacity; given a good offered load, a lightly-loaded network and some time, a TCP connection will always overtake a series of CoAP requests.

However, the [I-D.ietf-core-block] protocol can be used by inventive clients to emulate TCP slow start. E.g., a client can do a request for block 0, and, if a response comes back without a loss, it can fire off the requests for block 1 and block 2 at the same time, etc., using each response in a similar way that TCP would clock its data segments based on ACKs, waiving NSTART. Similar approaches may work to increase channel utilization for any other REST usage that requires multiple requests.

Clearly, the slow start period MUST terminate on the first loss/ retransmission. How exactly the congestion window is to be

maintained after that (a "congestion avoidance period" for CoAP) is a subject for further study. See also [I-D.mathis-tcpm-tcp-laminar] for fresh approaches to maintaining the necessary variables in TCP. Another alternative would be an implementation that emulates [RFC5348].

6. Changes Planned for Base Specifications

[I-D.ietf-core-coap]:

1. Change SHOULD in second paragraph of [I-D.ietf-core-coap] 4.7 to MUST; define protocol parameter NSTART.

2. Add reference to (and/or cite) [RFC5405] guideline about combining congestion control state for a destination; clarify its meaning for CoAP using the definition of an endpoint.

3. Add a mechanism of decaying outstanding transactions at a rate of about 7 B/s.

4. Add default "Leisure" (response dithering windows) of 10 seconds to [I-D.ietf-core-coap] 8.2, as well as n admonition for a client not to use multicast requests when such a default window would be way off.

[I-D.ietf-core-observe]:

1. servers SHOULD NOT send more than one NON notification every 3 seconds to an endpoint on average ([RFC5405] section 3.1.2); define protocol parameter MAXNONRATE.

2. servers SHOULD NOT send NON messages while waiting for CON messages to be acknowledged (however, CON retransmissions should send the new resource state if it has changed since, see [I-D.ietf-core-observe] section 4.5).

3. require sending a CON message at least every 24 hours before continuing with NON messages.

4. consider increasing MAXNONRATE for servers that check the client that rarely (e.g., to the rate at which TCP sends into a very lossy channel, e.g., 7 B/s).

Additional changes have been made to limit the leeway that implementations have in changing the CoRE protocol parameters; these changes are already gathered in Section 4.8 of [I-D.ietf-core-coap] and will not be repeated here.

7. IANA Considerations

This document makes no requirements on IANA. (This section to be removed by RFC editor.)

8. Security Considerations

(TBD. The security considerations of, e.g., [RFC2581], [RFC2914], and [RFC5405] apply. Some issues are already discussed in the security considerations of [I-D.ietf-core-coap].)

9. Acknowledgements

The first document to examine CoAP congestion control issues in detail was [I-D.eggert-core-congestion-control], to which this draft owes a lot.

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions that this draft attempts to answer.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

10.2. Informative References

[GF] Bormann, C., "Garrulity and Fluff", IAB Smart Object Workshop, 2011, <http://www.iab.org/wp-content/IAB-uploads/2011/04/Bormann.pdf>.

[HTTPBISc715] "Changeset 715", October 2009, <http://trac.tools.ietf.org/wg/httpbis/trac/changeset/715>.

[HTTPBISt131] "increase connection limit", HTTPBIS ticket #131, closed 2009-12-02, September 2008, <http://trac.tools.ietf.org/wg/httpbis/trac/ticket/131>.

[I-D.eggert-core-congestion-control] Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.

[I-D.ietf-core-block] Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-08 (work in progress), February 2012.

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-11 (work in progress), July 2012.

[I-D.ietf-core-observe] Hartke, K., "Observing Resources in CoAP",

draft-ietf-core-observe-05 (work in progress), March 2012.

[I-D.ietf-httpbis-p1-messaging]
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part 1: Message Routing and Syntax"", draft-ietf-httpbis-p1-messaging-20 (work in progress), July 2012.

[I-D.ietf-tcpm-initcwnd]
Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", draft-ietf-tcpm-initcwnd-04 (work in progress), June 2012.

[I-D.mathis-tcpm-tcp-laminar]
Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", draft-mathis-tcpm-tcp-laminar-01 (work in progress), July 2012.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

[RFC4336] Floyd, S., Handley, M., and E. Kohler, "Problem Statement for the Datagram Congestion Control Protocol (DCCP)", RFC 4336, March 2006.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org