<https://helda.helsinki.fi>

Performance Evaluation of Constrained Application Protocol over TCP

Järvinen, Ilpo Johannes

IEEE
2019

Järvinen , I J , Pesola , L M A , Raitahila , I K , Cao , Z & Kojo , M P I 2019 , Performance Evaluation of Constrained Application Protocol over TCP . in 2018 IEEE 88th Vehicular Technology Conference : VTC-Fall . , 8690641 , IEEE Vehicular Technology Conference , IEEE , IEEE Vehicular Technology Conference , Chicago , Illinois , United States , 27/08/2018 . <https://doi.org/10.1109/VTCFall.2018.8690641>

<http://hdl.handle.net/10138/325778>
<https://doi.org/10.1109/VTCFall.2018.8690641>

unspecified
acceptedVersion

Downloaded from Helda, University of Helsinki institutional repository.

This is an electronic reprint of the original article.

This reprint may differ from the original in pagination and typographic detail.

Please cite the original version.

Performance Evaluation of Constrained Application Protocol over TCP

Ilpo Järvinen*, Laura Pesola*, Iivo Raitahila*, Zhen Cao[†] and Markku Kojo*

*University of Helsinki

[†]Huawei

Email: ilpo.jarvinen@helsinki.fi, laura.pesola@helsinki.fi, iivo.raitahila@helsinki.fi,
zhen.cao@huawei.com, markku.kojo@cs.helsinki.fi

Abstract—The Constrained Application Protocol (CoAP) is specifically designed for constrained IoT devices and is being rapidly deployed for the communication needs of the IoT devices. CoAP has been specified with its own congestion control algorithms because it runs on top of UDP that does not include any congestion control measures. These algorithms aim at taking into account the specific needs of the IoT communication. The need of running CoAP also over TCP has arisen recently and is expected to be increasingly deployed alongside with CoAP over UDP. To understand the benefits and shortcomings of both CoAP over TCP and CoAP over UDP, we run an extensive set of experiments in different network settings and compare the performance of CoAP over TCP to the existing congestion control algorithms for CoAP over UDP. Our results reveal that even though CoAP over TCP has its known limitations it scales well and performs even better than expected in certain wireless settings that CoAP over UDP algorithms are specifically designed for, often even outperforming CoAP over UDP.

I. INTRODUCTION

Billions of Internet of Things (IoT) devices are expected to communicate over the Internet in the near future. Reliable communication with IoT devices commonly occurs as request-response pairs. This kind of communication, like any communication over the Internet, must be sufficiently congestion controlled. In IoT communication, however, many exchanges are short which is challenging for congestion control. The congestion control algorithm has to acquire enough state from the limited available information to prevent congestion from persisting as persistent congestion would cause severe performance problems. In addition, the main connectivity medium for IoT devices are wireless links because using wires would be impractical due to large number of devices and challenging deployment locations. Wireless links, in turn, differ quite significantly from their wired counterparts. Link errors due to corruption are much more frequent phenomena on wireless links than on wired ones. Designing congestion control for such link environments is even more challenging because the measures for effective congestion control and efficient recovery from link errors are contradictory.

The Constrained Application Protocol (CoAP) [1] is specifically designed for constrained IoT devices and is being rapidly deployed for the communication needs of the IoT devices. CoAP has been specified with its own congestion control algorithms because it runs on top of UDP that does not include

any congestion control measures. The base specification of CoAP includes only very limited congestion control based on retransmission timer backoff. CoAP Simple Congestion Control/Advanced (CoCoA) [2] aims to improve the CoAP congestion control algorithm such that persistent congestion is not caused by the CoAP traffic. It also aims at taking into account the specific needs of the IoT communication, particularly the operation over wireless links.

While CoAP has originally been designed to operate on top of UDP, the need of running CoAP also over TCP has arisen recently because some networks rate limit or even completely block UDP packets. UDP traffic may also have issues with Network Address Translation (NAT) traversal. Therefore, CoAP over TCP [3] has been specified and it is expected to be increasingly deployed alongside with CoAP over UDP. However, studies on the performance of the CoAP over TCP are very limited.

In this paper we run an extensive set of experiments to understand the benefits and shortcomings of both CoAP over TCP and CoAP over UDP in different network settings and compare the performance of CoAP over TCP to the existing congestion control algorithms for CoAP over UDP. To our knowledge, this is the first performance study on CoAP that includes CoAP over TCP to extensively explore its performance. Our results indicate that even though CoAP over TCP has its known limitations, in particular when new TCP connections need to be frequently established, it scales well to high levels of congestion and performs surprisingly well over error-prone wireless links, while the CoAP over UDP algorithms often yield suboptimal performance even though they are specifically designed for the challenging wireless communication in IoT environments.

II. BACKGROUND AND RELATED WORK

The Constrained Application Protocol (CoAP) [1] is aimed for constrained devices that commonly do not exchange a large amount of information at each time.

TCP has long been the most widely used transport protocol in the Internet and congestion control in the Internet relies on TCP to a large extent. The CoAP protocol runs on top of UDP and has by default extremely basic congestion control based on an initial retransmission timeout (RTO) that is randomized

between two and three seconds for the original transmission of a message and a binary exponential backoff of the RTO timer for retransmitted messages (we call this congestion control “Default CoAP” in this paper). No RTT estimation is performed by Default CoAP, meaning that each new message exchange starts with the initial RTO value.

CoAP Simple Congestion Control/Advanced (CoCoA) [2] introduces a new RTO mechanism for CoAP over UDP that is based on the TCP RTO computation algorithm [4] but runs two RTO estimators: a strong RTO estimator (E_{strong}) that uses the ACKs of the original transmissions to measure unambiguous RTT samples and a weak RTO estimator (E_{weak}) that uses the ACKs arriving after retransmissions, yielding ambiguous RTT samples. If the message is retransmitted more than two times, CoCoA skips the weak RTO estimator update. The strong RTO estimator is computed just like the TCP RTO but for the weak RTO estimator the RTT variation multiplier of the TCP RTO algorithm, the factor K , is set to 1 instead of 4.

The overall RTO for CoCoA is an exponentially weighted moving average updated each time a new RTT sample is taken and either E_{strong} or E_{weak} is updated. It is computed from the estimator with which the most recent RTT sample was taken using either equation 1 or 2:

$$RTO = 0.25 * E_{weak} + 0.75 * RTO \quad (1)$$

$$RTO = 0.5 * E_{strong} + 0.5 * RTO \quad (2)$$

In addition, CoCoA modifies the RTO backoff logic by using a variable backoff factor instead of a fixed backoff factor of 2 when the RTO timer expires. For each retransmission, CoCoA multiplies the RTO timer by 3 when the RTO is below 1 s and by 1.5 when the RTO is above 3 s. When the RTO is between 1 s and 3 s, the backoff factor is 2.

The congestion control of CoAP over UDP has been evaluated in several studies [5], [6], [7], [8]. In a recent study [9] both Default CoAP and CoCoA have been found to be incapable of coping with high level of congestion over a bufferbloat [10] network path. In [11] CoAP message exchanges using CoAP over TCP have been found to yield notably higher latency than those using CoAP over UDP.

III. TEST SETUP

A. Network

In our experiments multiple IoT devices communicate with a fixed host over a constrained link as shown in Figure 1. The Netem network emulator in the Linux kernel is used for emulating the constrained link that has the characteristics of an asymmetric, wireless link with a data rate of 30 kbps

Fig. 1: The test setup

and 60 kbps for downstream and upstream, respectively. The downstream link delay is 400 msecs and upstream delay is 200 msecs. The link MTU is set to 296 bytes.

We include a router queue with different buffer sizes in the front of the bottleneck link in order to experiment with the effect of a small buffer as well as larger buffer sizes representing different levels of bufferbloat that is typical in the various access networks deployed in the Internet today. The small buffer size is 2500 bytes that approximately equals to the bandwidth-delay product of the link and the larger sizes causing bufferbloat are 28200 bytes and 1410000 bytes, the latter representing an “infinite” buffer size. The delay for the rest of the path between the last-hop router and fixed host is set to 10-20 msecs with random variation.

To emulate random packet losses on the constrained wireless link we employ a two-state model where a good link state and bad link state alternate in short intervals. An exponential distribution is used for the state lengths. The mean length is 400 msecs and 100 msecs for good and bad states, respectively. The distributions are truncated at both ends; the good state between 60 msecs and 1200 msecs, and the bad state between 30 msecs and 300 msecs. We implement two different packet-error rate profiles: *medium* and *high*. During the bad state the packet-error rate is 80% for the high profile and 50% for medium profile, while the packet-error rate of the good state is 2% for the high profile and 0% for the medium profile. This yields an average packet drop rate of approximately 10% for the medium and 18% for the high error-rate profile.

B. CoAP Congestion Control Variants

We use Default CoAP as per RFC7252 [1], CoCoA that we implemented as per Internet-draft [2], and CoAP over TCP [3] on top of Linux TCP implementation. All CoAP variants under study are implemented in libcoap [12]. The CoAP protocol limits the maximum number of retransmissions to four by default [1]. We increase MAX_RETRANSMIT parameter to 20 to prevent a situation where retransmission of a message would be aborted during heavy congestion or frequent random losses, which could distort the results by producing uneven traffic load between the test cases. Similarly, limit for TCP SYN and SYN/ACK retries are increased to 40.

CoCoA is specified to truncate the backed off RTO at 32 seconds [2]. For Default CoAP we use 60 seconds backoff limit because we want to avoid unnecessarily long retransmission timeouts. For TCP, a maximum RTO value of 120 seconds is used as that is the default of the Linux TCP implementation. In addition, Linux TCP implementation is modified to send delayed ACKs with a constant 200 msecs timer and to use initial RTO of 2 seconds. Both CoAP over UDP congestion control variants use RTO dithering resulting in an initial RTO of 2-3 seconds.

C. Work Load

Two types of workload clients are used: *continuous* and *random*. A *continuous* client exchanges request-response pairs one by one until 50 pairs have been successfully exchanged.

TABLE I: Flow completion times with one and 50 clients

Client	n	Buffer	Cong. control	10th	25th	Median	75th	90th
Continuous	1	2500B	Default CoAP	33.003	33.003	33.003	33.003	33.003
Continuous	1	2500B	CoCoA	33.002	33.002	33.003	33.003	33.003
Continuous	1	2500B	CoAPoverTCP	33.208	33.208	33.208	33.208	33.208
Random	1	2500B	CoAPoverTCP	43.432	45.099	47.093	47.746	47.928
Continuous	50	2500B	Default CoAP	61.250	61.500	62.001	63.342	63.767
Continuous	50	2500B	CoCoA	61.275	61.525	62.001	63.347	63.767
Continuous	50	2500B	CoAPoverTCP	62.102	63.257	65.032	67.760	72.684
Random	50	2500B	CoAPoverTCP	83.196	86.248	88.433	90.463	93.199
Continuous	50	infinite	Default CoAP	62.177	62.377	62.690	63.003	63.178
Continuous	50	infinite	CoCoA	62.177	62.377	62.690	63.003	63.179
Continuous	50	infinite	CoAPoverTCP	69.382	69.570	69.868	70.184	70.383
Random	50	infinite	CoAPoverTCP	85.789	87.233	90.086	90.846	91.259

In the case of CoAP over TCP, all clients connect to the server before any requests are sent. That is, neither TCP three-way handshake nor the exchange of the CSM messages is included in the measured time. A *random* workload client comprises of a sequence short-lived clients that exchange a small random number (from 1 to 10) of request-response pairs each until 50 pairs altogether have been successfully exchanged. After each short-lived client the TCP connection is closed and a new TCP connection is established for the next short-lived client or, in the case of UDP, the CoAP state is reset. This emulates a situation where a single IoT device participates in exchanging only a few request-response pairs but is immediately followed by another device.

With Default CoAP the distinction between a continuous and random client does not make any difference as with it RTO values are not affected by the previous CoAP messages.

In order to introduce different levels of congestion, we vary the number of simultaneous clients from 1 to 400 that is large enough to cause severe congestion. Each test case over an error-free link is replicated 20 times and over an error-prone link 40 times.

IV. RESULTS

We present the measurement results with small request-response exchanges, having 60 bytes of response payload. We experiment with a varying number of simultaneous clients communicating with a server over a shared bottleneck link.

A. Results over an Error-free Link

The traffic over an error-free bottleneck link is subject to congestion-related losses only.

1) *Results with 1 and 50 Clients:* Table I shows the median, 25th and 75th percentile (quartiles), and 10th and 90th percentile of the flow completion time (FCT) for 1 and 50 simultaneous clients. The FCT is the elapsed time for a client to complete 50 message exchanges. When only one client is transmitting, it exchanges 50 messages in 33.003 to 33.208 seconds. A single request-response exchange is transferred at a time leaving the router buffer empty. No congestion occurs and the FCT is limited only by the round-trip time (RTT) of the network path that is around 660 msec with one client. This is the best case scenario with ideal performance in the given environment, setting a baseline for other scenarios. There is no difference in results between Default CoAP and CoCoA, because no retransmissions are made.

With a continuous client the FCT for CoAP over TCP is around 200 milliseconds (less than 1%) longer than for the

UDP counterparts. This is only because of the higher header overhead with TCP (the three-way handshake is not included in the FCT with continuous clients).

With Default CoAP and CoCoA, the FCT for a single random client does not differ from that of the continuous client nor are there notable differences in the FCTs for 50 continuous and random clients. Therefore, the results for Default CoAP and CoCoA with random clients are omitted from the table. However, the FCT for a single random client using CoAP over TCP is notably longer than that of the continuous client. Every time a new short-lived client is started, the client has to establish a new TCP connection and send a CSM message. The TCP three-way handshake takes an additional RTT and another RTT is spent with the CSM message due to the TCP Nagle's algorithm [13], resulting in 41.8% higher median FCT for a random client using CoAP over TCP compared to that of a continuous client.

With 50 simultaneous clients the median FCT increases for all variants being between 62 and 65 secs continuous clients using the small router buffer. With random clients and the small router buffer, the median FCT increases even up to 88.433 secs for CoAP over TCP.

The major reason for the increased FCT compared to the one client case is in increased queueing delay as messages start to pile up in the router buffer and thereby increase the perceived RTT. In addition, some congestion drops start to occur with the small buffer. However, only a very few messages are dropped with Default CoAP and CoCoA in the beginning and only for some of the simultaneous clients. Retransmitting the few dropped messages increases slightly the FCT for those clients and becomes visible at the upper percentiles for Default CoAP and CoCoA.

With TCP, the FCTs increase more than with Default CoAP and CoCoA. The median FCT for the continuous clients using CoAP over TCP is approx. 3 seconds (about 5%) longer than that of Default CoAP and CoCoA. The reason for the higher median FCT with CoAP over TCP is the larger header overhead that results in longer queueing delay. Effectively there are roughly the same number of larger CoAP over TCP messages queued as there are smaller CoAP over UDP messages because more CoAP over UDP messages than CoAP over TCP messages fit into the network over a single RTT. In addition, the larger TCP header overhead allows a bit less response messages to fit into the small router buffer with CoAP over TCP than with Default CoAP or CoCoA, resulting in some additional packet losses that CoAP over TCP has to recover from. This becomes visible as a more notable increase in FCT towards the upper percentiles.

The infinite router buffer can absorb more packets eliminating all packet losses. This increases the queueing delay because the packets dropped with the 2500 bytes buffer now fit into the buffer, but at the same time FCTs decrease towards the upper percentiles as no packet losses need to be recovered. In addition, the FCTs become more stable compared to the case with the small buffer. The median FCT for continuous clients using CoAP over TCP is approximately 7.2 seconds

Fig. 2: Flow completion time with 100 clients

(about 11%) longer than that of Default CoAP and CoCoA. This difference, again, is due to the larger header overhead with TCP and longer queueing delay. The TCP packets that were dropped by the small buffer now fit into the larger buffer and thereby further increase the queueing delay for CoAP over TCP.

With 50 random clients the FCTs for CoAP over TCP with both buffer sizes are notably higher than with continuous clients as expected. Not only does the TCP connection establishment for each new short-lived client require an additional time-consuming RTT, but the TCP three-way handshake is also subject to packet losses and retransmissions with the small buffer and subject to longer queueing delay with the infinite buffer.

2) *Results with 100 Continuous Clients:* The median, quartiles, and 10th and 90th percentile of the flow completion times for 100 simultaneous clients are shown in Figure 2. With 100 simultaneous clients the median FCTs increase substantially compared to the 50 simultaneous clients case. This is due to increased congestion that results in more packet losses with the smaller router buffer and longer queueing delay with the infinite router buffer.

With continuous clients and the 2500 bytes router buffer the median FCT is around 101.6 seconds for Default CoAP and around 104.7 seconds for CoCoA while TCP over CoAP yields a shorter median FCT being around 86.7 seconds. All three congestion control variants react to congestion by backing off the retransmission timer value when retransmitting. This allows exchanging most of the messages without retransmissions, while those clients that need retransmissions suffer from long FCTs. In some cases when a response message hits a full router queue and becomes dropped, it results in an unfortunate iteration of retransmissions where the retransmitted response message also hits the full queue. This may get repeated a number of times and each time the timer is exponentially backed off. As a result, the FCT for these unlucky clients becomes notably increased and is visible in the upper FCT percentiles of Figure 2. This phenomenon represents the classical lock-out behaviour [14] on a congested tail-drop router queue.

With the 2500 bytes buffer the median and lower percentile FCTs for TCP are shorter than those of Default CoAP and CoCoA. The reason is mainly because TCP responds to congestion more effectively than the other algorithms by keeping the backed off timer value according to Karn's algorithm [15] until a new data is acknowledged without retransmission. This decreases the congestion level and allows TCP to complete with a lower number of retransmissions than Default CoAP and CoCoA. Consequently, those CoAP over TCP clients that do not back off are allowed to make faster progress with lower congestion level and those CoAP over TCP clients that do back off suffer from longer FCTs than the CoAP over UDP variants as can be seen in the upper percentiles of Figure 2.

Instead of keeping the backed off RTO value Default CoAP restores the initial RTO value and CoCoA the current RTO after receiving a response to a retransmitted request, both thereby sending the next message with a lower timer value than CoAP over TCP even though there is no indication that congestion has cleared. This increases congestion losses and number of retransmissions for those algorithms.

With 100 continuous clients and the infinite router buffer, the buffer can absorb more packets eliminating all losses. Again, the larger buffer increases the queueing delay and results in significantly longer median FCTs for all congestion control variants. Interestingly enough, the positions of CoAP over TCP and CoCoA change now with the larger router buffer. TCP becomes again slower than CoCoA due to the larger header size and thereby also longer queueing delay. However, Default CoAP slows down dramatically. With 100 clients and the infinite buffer, the queueing delay increases such that the RTT becomes extended well beyond 2 seconds. It means that with all congestion control variants many of the early message exchanges that use an RTO value close to the initial RTO of 2 seconds and hit the long router queue inevitably encounter a spurious retransmission timeout. These unnecessary retransmissions increase the queue delay and consequently the RTT, resulting in more spurious RTOs. TCP and CoCoA are able to soon adjust the RTO values beyond the actual RTT and unnecessarily retransmit only a few messages per client. CoCoA avoids many spurious RTOs because it is better guarded against them than TCP as its RTO is dithered to a larger retransmission timer value, while TCP uses its calculated RTO value directly for the retransmission timer. Default CoAP, however, uses the initial RTO of 2-3 seconds for each original message transmission, resulting in one unnecessary retransmission for almost all request messages. Therefore, the FCT for Default CoAP is nearly twice as long as for CoCoA.

3) *Results with 100 Random Clients:* With 100 random clients and the smallest buffer the results with Default CoAP and CoCoA are still similar to those with continuous clients, but the overhead of the TCP three-way handshake with retransmitted SYN/ACK segments results in notably longer FCTs for CoAP over TCP as expected. With the infinite buffer, the FCT for CoCoA and CoAP over TCP increases compared to the FCT with continuous clients, because each new short-

Fig. 3: Flow completion time with continuous clients

lived client starts with the initial RTO value that is often lower than the real RTT, resulting in spurious RTOs.

4) *Results with 200 and 400 Continuous Clients:* Figure 3 shows the flow completion times for 200 and 400 simultaneous clients of continuous type. The higher level of congestion results in further increase in the number of packet losses or in the queueing delay depending on the router buffer size, and the FCTs are further extended as expected.

TCP over CoAP keeps its position ahead of the CoAP over UDP variants with the small router buffer that causes a notable number of packet losses. The median FCT with the 2500 bytes buffer and 200 clients is about 16.6 seconds (approx. 9%) shorter for the TCP clients than that of Default CoAP and about 27.2 seconds (approx. 13.7%) shorter than that of CoCoA. Accordingly with 400 clients the median FCT for TCP is about 70.1 seconds (approx. 18%) shorter than that of Default CoAP and about 88.8 seconds (approx. 21.8%) shorter than that of CoCoA. With larger buffers given that the buffer size in question results in a notable number of losses, namely the buffer sizes up to 28200 bytes with 400 clients, TCP is also a bit faster completer than CoCoA. The reason why TCP is faster completer with smaller buffers is the same as with the small 2500 bytes buffer and 100 clients, TCP being more responsive to congestion.

When there are little or no packet losses with larger buffer sizes, namely with the infinite buffer and the 28200 bytes buffer with 200 clients, the queueing delay increases alongside the increase in the buffer size and the number of clients. The larger queueing delay inherently increases the FCT for all clients. Both CoCoA and CoAP over TCP consistently measure the RTT and are able to adjust their RTO values. Both variants converge to the prevailing RTT and no more unnecessary retransmissions are needed after the early message exchanges. Hence, the larger header overhead with CoAP over TCP mainly determines the differences in the FCTs between CoAP over TCP and CoCoA. In contrast, while

Fig. 4: Unnecessary retransmissions per client for 400 clients

Default CoAP manages reasonably well the large number of packet losses that occur with 200 and 400 clients and the small router buffer it indicates even more unsustainable behaviour with the larger buffer sizes. The long queueing delay results in a huge number of spurious RTOs with Default CoAP because Default CoAP is not able to adjust its RTO. It is not enough that Default CoAP backs off its timer on retransmissions, because it reverts the initial RTO of 2-3 seconds for each new message, resulting to another round of spurious RTOs until the timer is again backed off beyond the prevailing RTT. These spurious RTOs fill up the large buffers with unnecessary retransmissions that further increase the round-trip time and most of the capacity of the bottleneck link is wasted in exchanging unnecessarily retransmitted messages. Figure 4a shows the number of unnecessary retransmissions for different CoAP variants with 400 continuous clients. Default CoAP retransmits most of its requests unnecessarily four times. This behaviour closely resembles congestion collapse [13], because the network makes only little forward progress. Instead, it delivers unnecessary retransmissions most of the time.

5) *Results with 200 and 400 Random Clients:* The flow completion times for 200 random clients are shown in Figure 5a. With 200 random clients and the small 2500 bytes buffer Default CoAP still behaves similar to the continuous clients and so does TCP over CoAP, but the extra TCP connection establishments slow down CoAP over TCP notably as expected. CoCoA, in turn, has some problems of adjusting its RTO to a proper value and therefore it uses somewhat larger RTO values than Default CoAP; every now and then when a CoCoA client retransmits it gets a weak RTT sample that inflates the RTO or when it gets an unambiguous RTT sample, the RTO calculation of CoCoA is too slow to converge towards the actual RTT of around 1.3 seconds. Instead, the RTO of CoCoA stays mostly well above 2 seconds. Hence, on the average the RTO timer expires a bit later for CoCoA than the initial RTO of 2-3 seconds for Default CoAP, resulting

Fig. 5: Flow completion time with random clients

in longer recovery time for dropped messages. As a result, CoCoA performs somewhat worse than Default CoAP. The same phenomenon happens also towards the beginning of the flow with continuous CoCoA clients, but in the later phases the continuous CoCoA clients are able to converge the RTO to values below 2 seconds which somewhat compensates the longer RTOs in the beginning of the flow and makes continuous clients to complete a bit faster than random clients.

With 200 random clients and larger buffers CoCoA continues to complete faster than CoAP over TCP, but the FCTs for CoCoA are much higher than with continuous clients. This is due to the increasing number of unnecessary retransmissions as CoCoA is not able to adjust its RTO timely.

The flow completion times for 400 random clients are shown in Figure 5b. With the small buffer Default CoAP still performs fastest, while CoCoA suffers from the same problems with its RTO adjustment as with 200 clients. Similarly, the extra TCP connection establishments still slow down CoAP over TCP.

With 400 random clients and the 28200 bytes buffer the results follow the similar trend as with smaller number of clients. However, with the infinite buffer the results turn around compared to the results with smaller number of clients as well as compared to the results with continuous clients: CoAP over TCP completes notably faster than the other two as the FCT for CoCoA explodes being even longer than that of Default CoAP. For Default CoAP it takes roughly twice as long to complete than for CoAP over TCP and for CoCoA it takes approx. 134% longer time to complete than for CoAP over TCP. CoCoA is incapable of timely adjusting its RTO to the high RTT level due to the bufferbloomed queueing. Therefore, it ends up using RTO values close to that of Default CoAP. This results in even higher number of unnecessary retransmissions than with Default CoAP because CoCoA retransmits even more aggressively than Default CoAP as can be seen in Figure 4b.

Fig. 6: Flow completion time over an error-prone link

CoCoA unnecessarily retransmits each message 5-6 times, while Default CoAP represents the same behaviour as with continuous clients. The reason for more aggressive retransmissions is that CoCoA mostly uses the variable backoff factor of 1.5, while Default CoAP uses the more conservative backoff factor of 2. In contrast, CoAP over TCP unnecessarily retransmits only a CoAP few messages. It, however, inevitably retransmits the TCP SYN/ACK segments a few times as well as some number of the CSM messages that it exchanges in the beginning of each TCP connection. Altogether, the number of unnecessarily retransmitted messages is clearly lower for CoAP over TCP than for the CoAP over UDP variants as shown in Figure 4b. This indicates that CoAP over TCP scales relatively well with the increasing level of congestion, while neither Default CoAP nor CoCoA is able to scale properly with increasing congestion in a bufferbloomed network environment.

B. Results over an Error-prone Link

Figure 6 shows the median, quartiles, and 10th and 90th percentile of the flow completion time for 10 simultaneous clients. When the packet-error rate increases, the FCT starts to grow with all CoAP variants as expected.

1) *Results with Continuous Clients:* When using continuous clients and the medium error rate the median FCT for CoCoA is 12.4% and for Default CoAP 38.4% longer than that of CoAP over TCP, while with the high error rate the median FCT for CoCoA is 12.9% and for Default CoAP 35.0% longer than that of CoAP over TCP.

With continuous clients, the reason for CoAP over TCP performing better than the other two algorithms is in the accuracy of the RTO computation that allows CoAP over TCP to use RTO values closer to the actual round-trip times and thereby react faster to packet losses. The RTO computation of CoCoA converges slower than that of TCP because CoCoA uses the additional weight of 0.5 when applying the strong RTT estimate. In addition, CoCoA takes ambiguous weak RTT measurements also into account in its RTO computation,

resulting in higher RTO values because each weak RTT sample unnecessarily inflates the overall RTO. Default CoAP reuses the initial RTO value with each message and is unable to converge its RTO value towards the prevailing RTT, which is around 660 msecs.

In addition, with the high error rate profile CoAP over TCP has most deviation towards the upper percentiles. This is mainly due to the Karn's algorithm where TCP, after retransmitting, keeps the backed off RTO value and uses it for the next message, while Default CoAP and CoCoA restore a more aggressive RTO value after repairing a loss. Hence, when consecutive messages get dropped, TCP uses higher, backed off RTO values than Default CoAP and CoCoA. This gives unjustified benefit for the UDP-based algorithms that trade their performance in the presence of congestion to obtain efficient recovery from random losses. As a result, keeping the backed off RTO value allows TCP to respond appropriately to heavy congestion unlike the CoAP over UDP variants that suffer from severe performance problems under heavy congestion as discussed earlier. And, despite of the unjustified benefit for the CoAP over UDP variants, CoAP over TCP recovers more efficiently from random losses than the CoAP over UDP variants.

2) *Results with Random Clients*: The most notable difference in the results with random clients compared to the results with the continuous clients is that CoAP over TCP has the longest median FCT with all error rate profiles. This stems from the additional overhead of the TCP connection establishments just like in the tests over the error-free link discussed earlier. The overhead with the TCP three-way handshake becomes significant over the error-prone link because also the TCP SYN and SYN/ACK as well as segments carrying CSM messages are subject to drops and retransmissions thereby increasing the FCT for CoAP over TCP.

However, when the error rate increases the difference in FCT is not that drastic as on the error-free link where the median FCT of CoAP over TCP is 43.3% longer than that of the other algorithms. With the high error rate profile the median FCT for CoAP over TCP is only 36.8% longer than that of CoCoA.

C. Summary of the Results

Our results indicate that CoAP over TCP scales even to heavy congestion. Default CoAP and CoCoA that do not retain the backed off RTO for the next message, however, are potentially even dangerous to the stability of the Internet and may cause congestion collapse, and therefore their use should be considered carefully. Not retaining the backed off RTO also implements an unjustified performance optimization for the CoAP over UDP variants in scenarios with random losses because it may under heavy load trigger congestion collapse that should be avoided at all cost. Despite the unfair advantage the CoAP over UDP variants have over CoAP over TCP, the accurate RTO calculation with TCP allows recovering more efficiently from random losses.

CoAP over TCP is not very well suited for short-lived CoAP clients because of the overhead in creating new TCP connections frequently.

V. CONCLUSIONS

In this paper, we set up an emulated IoT network environment and run an extensive set of experiments with small CoAP request-response exchanges in various network settings comparing the performance of CoAP over TCP to the existing congestion control algorithms for CoAP over UDP.

Our results show that, in general, CoAP over TCP performs equally well or better than the CoAP over UDP variants unless new TCP connections are frequently established, which is a known shortcoming of TCP for typical IoT communication patterns. We confirm that CoAP over TCP scales well and responds appropriately to heavy congestion, while the existing CoAP over UDP algorithms do not properly respond to high level of congestion when large bufferbloat router buffers are present. In the presence of random losses that are typical for wireless links, CoAP over TCP is competent in recovering the losses and even outperforms the CoAP over UDP variants when long-lived TCP connections can be used.

REFERENCES

- [1] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014.
- [2] C. Bormann, A. Betzler, C. Gomez, and I. Demirkol, "CoAP Simple Congestion Control/Advanced," Internet Draft draft-ietf-core-cocoa-03, Feb. 2018, Work in progress.
- [3] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets," RFC 8323, Feb. 2018.
- [4] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298, Jun. 2011.
- [5] A. Betzler, C. Gomez, I. Demirkol, and J. Paradelles, "CoAP Congestion Control for the Internet of Things," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 154–160, Jul. 2016.
- [6] —, "CoCoA+: An Advanced Congestion Control Mechanism for CoAP," *Ad Hoc Networks*, vol. 33, pp. 126–139, 2015.
- [7] A. Betzler, C. Gomez, I. Demirkol, and M. Kovatsch, "Congestion Control for CoAP Cloud Services," in *Proc. 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sep. 2014, pp. 1–6.
- [8] I. Järvinen, L. Daniel, and M. Kojo, "Experimental Evaluation of Alternative Congestion Control Algorithms for Constrained Application Protocol (CoAP)," in *Proc. 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec. 2015.
- [9] I. Järvinen, I. Raitahila, Z. Cao, and M. Kojo, "Is CoAP Congestion Safe?" in *Proc. Applied Networking Research Workshop 2018 (ANRW'18)*, Jul. 2018.
- [10] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *ACM Queue*, vol. 9, no. 11, Nov. 2011.
- [11] F. Zheng, B. Fu, and Z. Cao, "CoAP Latency Evaluation," Internet Draft draft-zheng-core-coap-latency-evaluation-00, Jul. 2016, Work in progress.
- [12] "libcoap: C-Implementation of CoAP." [Online]. Available: <https://libcoap.net/>
- [13] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC 896, Jan. 1984.
- [14] F. Baker and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management," RFC 7567, Jul. 2015.
- [15] P. Karn and C. Partridge, "Improving Round-trip Time Estimates in Reliable Transport Protocols," in *Proc. ACM Workshop on Frontiers in Computer Communications Technology (SIGCOMM'87)*, Aug. 1987, pp. 2–7.