FASOR Retransmission Timeout and Congestion Control Mechanism for CoAP

Ilpo Järvinen*, Iivo Raitahila*, Zhen Cao† and Markku Kojo*

*University of Helsinki †Huawei

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

***Abstract*—The Constrained Application Protocol (CoAP) has been designed to be used on constrained devices such as Internet of Things (IoT) devices. The existing congestion control algorithms for CoAP have known shortcomings in addressing congestion and retaining a good level of performance when link errors occur. In this paper, we propose Fast-Slow RTO (FASOR) mechanism that takes into account special needs in wireless environments while still properly addressing congestion. We run a series of experiments to confirm that FASOR is able to successfully cope with challenging network conditions such as bufferbloat, high level of congestion, and high link-error rates unlike the default and CoCoA congestion control that have severe problems with bufferbloated congestion.**

I. Introduction

Congestion Control of TCP is the building block of the global Internet, ensuring reliable end-to-end communication, as well as peaceful operation of the network to avoid congestion collapse [1]. Internet TCP congestion control has matured after decades of experience learned from the network operation and application evolution. Internet of Things (IoT), which is expected to connect the next billions of devices together, confronts a similar set of problems but with additional challenges. Notably, the last-mile links are relatively prone to corruption, and packet loss rates due to link errors are much higher on them than on the well provisioned Internet links.

The Constrained Application Protocol (CoAP) [2] is the standardized way to provide end-to-end communication for IoT and it has been increasingly deployed. CoAP provides reliability and congestion control in the form of confirmable messages. However, the original specification in RFC 7252 provides only a very limited congestion control mechanism based on binary exponential retransmission timer backoff. CoAP Simple Congestion Control/Advanced (CoCoA) [3] aims to offer some improvements in order to help CoAP to recover from wireless losses more efficiently. However, a recent study [4] reveals that both default CoAP and CoCoA algorithms fail to properly control the congestion in a usual deep-buffered link (also known as bufferbloat [5]), because their Retransmission Timeout (RTO) mechanisms and backoff logics are not able to cope with cases when queuing delay bursts or is significantly above initial RTO.

With the above experience, we propose Fast-Slow RTO (FASOR) [6], an alternative RTO computation algorithm and backoff logic for congestion control. FASOR's design tackles two notable challenges of IoT congestion control with its novel design.

1) Cope with ambiguous RTT samples [7]. When retransmissions occur, it is difficult to match the arriving acknowledgement with the different copies of the outgoing request. Neglecting all ambiguous samples as specified by RFC 6298 [8] is a sacrifice of the opportunity to know the underneath features of the path. Instead, FASOR separates the RTO computation into two different categories, that is, employing a Fast RTO computation for unambiguous samples while introducing a Slow RTO to trace the ambiguous samples to cope with deep bufferbloat and heavy congestion. In this way, FASOR is able to reduce flow completion time under link errors and avoid introducing extra delay.

2) Tradeoff between aggressive and conservative retransmissions logic. Aggressive backoff risks into too many wasteful retransmission, while a conservative one wastes the opportunity to perform a quick transfer. FASOR is designed with a novel and self-adaptive retransmission timer backoff logic, consisting of three transitive states FAST/FAST_SLOW_FAST/SLOW_FAST each with different backoff logic adaptive to different network states. In this way, FASOR is able to take a risk properly and avoid additional power consumption caused by wasteful retransmissions.

We conduct an extensive evaluation of FASOR's performance under varying level of congestion, different buffer sizes, and link-error rates. We confirm that FASOR is able to cope with challenging network conditions. Slow RTO in FASOR successfully handles even the most extreme bufferbloat scenario, whereas the current CoCoA and Default CoAP congestion controls do not handle the retransmission timer backoff in a congestion safe manner. In addition to handling congestion properly, FASOR is able to outperform Default CoAP and CoCoA in link-error cases.

The rest of this paper is organized as follows. Section II explains the current state of the CoAP congestion control algorithms and Section III describes the proposed FASOR algorithm. Sections IV and V explain the performance evaluation environment and results. In Section VI we conclude this work.

II. Background and Related Work

The Constrained Application Protocol (CoAP) [2] is aimed for constrained devices that commonly do not exchange a large

amount of information at each time. Thus, small payloads are common to occur. Larger payloads can be delivered by splitting the payload into multiple blocks using the block-wise option [9] with which a request for the next block is sent only after the response to the previous one has been received.

The CoAP protocol 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) [3] specifies a more complex RTO calculation based on the TCP RTO computation algorithm [8] but runs two RTT estimators: a strong RTT estimator that uses the ACKs of the original transmissions to measure unambiguous RTT samples and a weak RTT estimator that uses the ACKs arriving after retransmissions, yielding ambiguous RTT samples. If the message is retransmitted more than two times, CoCoA skips the weak RTT estimator update. The strong RTT estimator is computed just like the TCP RTO but for the weak RTT estimator the RTT variation multiplier of the TCP RTO algorithm, the factor K, is set to 1 instead of 4. The overall RTO is an exponentially weighted moving average computed from the RTT estimator that made the most recent contribution as follows.

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

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

In addition, CoCoA modifies the RTO backoff logic of TCP 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 RTO is below 1 s and by 1.5 when RTO is above 3 s. When RTO is between 1 s and 3 s, the backoff factor is 2.

The congestion control of Default CoAP and CoCoA has been evaluated in different settings. The evaluation results indicate that CoCoA, in most cases, yields better performance than Default CoAP in terms of throughput and recovery from traffic bursts [10], [11] as well as in terms of latency [12].

A comparative performance study of the Default CoAP and CoCoA shows that CoCoA has higher throughput, shorter elapsed time, and a smaller number of retries compared to Default CoAP [13].

On the other hand, CoCoA has been shown to be somewhat more aggressive than Default CoAP at higher congestion levels [14].

A recent study [4] of CoAP congestion control shows that under a set of circumstances encompassing bufferbloat [5], which may well occur in real networks, neither Default CoAP nor CoCoA properly respond to congestion. At high congestion levels with a bufferbloated bottleneck buffer this even leads to congestion collapse [1] where unnecessary retransmissions consume most of the network resources.

III. Algorithm

FASOR (specified in [6]) is composed of three key components: Fast RTO computation, Slow RTO, and novel retransmission timer backoff logic. In addition, RTO values are dithered and the RTO value is upper-bounded by 60 seconds.

The Fast RTO computation is based on the TCP RTO computation [8] without minimum RTO bound. It is updated only with unambiguous RTT samples and therefore tracks closely the real RTT. The 1 second lower-bound for RTO is removed because with TCP RTO is only a backup mechanism for loss detection and lower latency for triggering loss recovery is possible using Fast Retransmit and Fast Recovery [15], whereas with CoAP RTO is the primary and only loss detection mechanism. Allowing lower RTO values than 1 second may improve latency for the environments where RTT is inherently less than 1 second. Initial RTO is set to two seconds and on the first unambiguous RTT sample, RTO is initialized with the newly acquired RTT sample like TCP does [8]. For faster convergence with short exchanges, the RTT variation initialization in FASOR uses a modified formula:

$$RTTVAR \leftarrow R/(2K) \quad (3)$$

where R is the RTT sample and K (RTT variation multiplier) is 4.

Slow RTO is analogous to Karn's algorithm that does not reduce a backed off RTO until an unambiguous ACK has been received for new data [7]. Slow RTO is measured from the original transmission of a CoAP request till the arrival of the ACK, regardless of the number of required retransmissions. In addition, Slow RTO is multiplied by a factor to allow some growth in load without making Slow RTO too aggressive (1.5 used in the measurements for this paper). Instead of keeping the backed off RTO with the next message exchange, we use Slow RTO as a key component for the FASOR backoff logic.

Because of a potentially high latency cost associated in using Slow RTO as the backed off RTO, FASOR uses a novel retransmission timer backoff logic to reduce the impact of Slow RTO for the cases when RTOs occur due to reasons unrelated to congestion. Hence, the retransmission backoff logic is a compromise between the cases with link errors that would benefit from always using the Fast RTO-based backoff and the cases with heavy congestion that would benefit from always using Slow RTO.

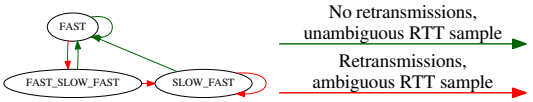

Fig. 1: FASOR state diagram

Figure 1 shows the FASOR states and state transitions for the FASOR backoff logic. It has three backoff states that are from the least conservative to most conservative: FAST, FAST_SLOW_FAST, and SLOW_FAST. Each time receiving an ACK, a FASOR sender decides in which state it performs the subsequent message exchange. After receiving

ACK without retransmissions, the FASOR sender always stays in or transits back to the FAST state. If an ACK is received only after retransmitting, FASOR downgrades into a more conservative state until it reaches the SLOW_FAST state where it stays as long as completing a message exchange requires retransmissions.

In each state, starting from the original transmission of a message, a different series of RTOs is used to back off RTO when retransmitting. The RTO timer values used in the backoff series of each state are as follows (Fast and Slow denote the value of Fast RTO and Slow RTO, respectively):

- **FAST**: Fast, Fast·2^1, Fast·2^2, ...
- **FAST_SLOW_FAST**: Fast, max(Slow, Fast·2), Fast·2^1, Fast·2^2, ...
- **SLOW_FAST**: Slow, Fast, Fast·2^1, Fast·2^2, ...

A FASOR sender starts in the FAST state and uses Fast RTO to arm the timer. If retransmissions are needed in the FAST state, the sender backs off Fast RTO until an ACK is received and then transits to the FAST_SLOW_FAST state for the next message exchange.

In the FAST_SLOW_FAST state, instead of using Slow RTO immediately, FASOR first probes the network using Fast RTO. If that probe is successful, FASOR does not need Slow RTO at all and instead concludes that RTOs in the FAST state likely did not occur due to heavy congestion but were more likely due to reasons unrelated to congestion. Only if the probe also results in an RTO, the second RTO in the FAST_SLOW_FAST state is armed using Slow RTO. In addition, once an ACK arrives, FASOR will also transit to the SLOW_FAST state where it uses Slow RTO immediately for the original transmission of the next message. This is intended to ensure that the sender can acquire an unambiguous RTT sample for the next message exchange even in a heavily bufferbloated environment.

The use of Slow RTO allows messages that the sender has potentially unnecessarily retransmitted to drain from the network. It also allows long enough time period to acquire an unambiguous RTT sample for updating Fast RTO with very high probability even if the network is heavily congested with a bufferbloated real RTT that is inflated beyond the current Fast RTO value. Hence, Slow RTO allows rapidly converging towards a safe operating point because draining duplicate copies from the network itself reduces the perceived RTT. We believe FASOR achieves a good balance in handling congestion and link errors. Even though FASOR uses an ambiguous RTT sample for Slow RTO, it is actually on the safe side because it never takes a too small ambiguous sample but uses the worst-case as the measurement base.

In order to improve the RTO computation we introduce a token version of the FASOR algorithm that supplements the RTO computation logic by making all RTT samples unambiguous. To distinguish whether an ACK arrives for the original transmission of a message or for a retransmission of it, an ordinal number of the message transmission is encoded into the CoAP token that may be included in a message exchange. We use one byte from the token space to encode the transmission number. The other endpoint echoes the token unmodified in the response as per the CoAP specification, thus no modification to the other endpoint is required. However, a token cannot be used with an Empty Acknowledgement (or Reset) message because the CoAP protocol specification does not allow adding a token to an Empty message. Therefore, the token-enhanced version of FASOR is useful only for the common case of a piggybacked CoAP response.

The client stores the timestamps of the original transmission and of each retransmission. When a response to the outstanding request arrives, the token value is inspected and the RTO is computed using the unambiguous RTT measurement from the timestamp of the corresponding transmission. This results in accurate RTO computation with only a small overhead.

IV. Test Setup

A. Network

The system under study consists of multiple IoT devices that communicate with a fixed host over an emulated constrained link. Netem included in the Linux kernel is used for the network emulation. The emulated network has the characteristics of an asymmetric, wireless link with a data rate of 30 kbps downstream and 60 kbps upstream, a one-way delay of 400 msecs downstream and 200 msecs upstream and an MTU of 296 bytes. The test environment is shown in Figure 2.

Three different values for the buffer size at the bottleneck are used: 2500 bytes (approximately the bandwidth-delay product of the link), 28200 bytes, and the "infinite" buffer of 1410000 bytes. The larger buffer values cause bufferbloat. 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 packet losses on the wireless link we use two different average packet error rate profiles: *medium* and *high*. We employ a two-state Markov model where a low-error state and an error-burst state alternate in suitably short intervals. During the error-burst states the error rate is 80% for high and 50% for medium. The average error rate of the low-error state is 2% for high and 0% for medium. This results in the average packet error rate being approximately 10% for medium and 18% for high link-error profile.

B. CoAP Congestion Control Variants

We use the libcoap [16] implementation of the CoAP protocol in the experiments. We compare the congestion control algorithms Default CoAP [2] as implemented in libcoap, CoCoA that we implemented as per the most recent Internet draft (version 03) [3], and FASOR. In addition, we include the congestion safe variant of CoCoA (CoCoA+FB1) that addresses the congestion collapse problem with it [4].

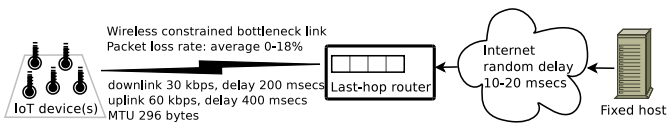

Fig. 2: The test setup

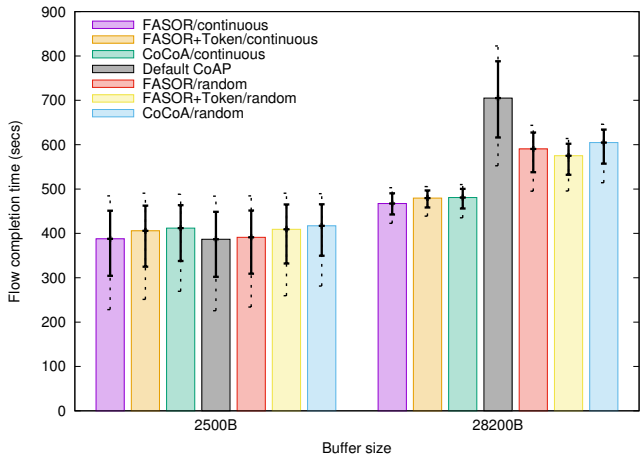

Fig. 3: Flow completion time with 400 flows using different congestion control algorithms

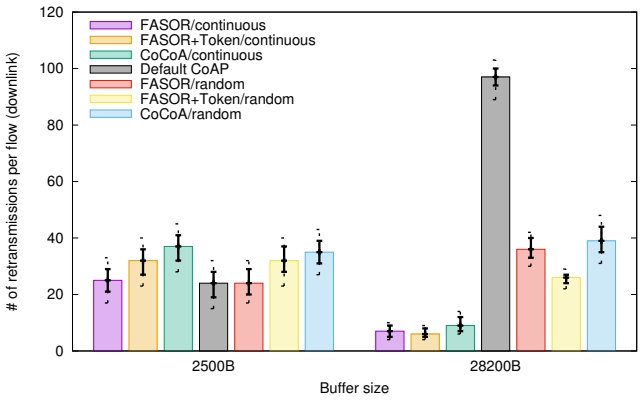

Fig. 4: Number of retransmissions per flow (over 400 flows) using different congestion control algorithms

We introduced certain changes to CoAP congestion control: MAX_RETRANSMIT is set to 20 and both EXCHANGE_LIFETIME and MAX_TRANSMIT_WAIT are adjusted accordingly [2]. The motivation for the changes is to prevent a situation where retransmission of a message would be aborted after the fourth retransmission (the default) during heavy congestion or frequent packet losses that may distort the results due to uneven load between the test cases.

All CoAP congestion control variants use an initial RTO of 2 seconds with dithering. FASOR implementation uses a 60 seconds maximum RTO. CoCoA [3] is specified to truncate the backed off RTO at 32 seconds. For Default CoAP we use 60 seconds backoff limit because no maximum value is defined and we want to avoid unnecessarily long retransmission timeouts. In addition, we do not implement the aging mechanism in CoCoA as it is intended to be applied only after idle periods that are not present in our workload.

C. Work Load

Two types of workload flows are used: *continuous* and *random*. For a *continuous* flow, a client exchanges CoAP request-response pairs until 50 pairs have been successfully exchanged. A *random* flow consist of a series of short-lived CoAP clients each generating a random number (between 1 and 10) of CoAP request-response pairs until altogether 50 pairs have been successfully exchanged. After each short-lived client terminates, the CoAP state is reset. This emulates a situation where a single device exchanges only a few request-response pairs with the server but is immediately followed by another device. With Default CoAP the distinction between continuous and random flows does not make any difference as with it RTO values are not affected by the previous CoAP messages.

The number of concurrent flows varies from 10 to 400. We run the error-free test cases with 20 replications and the link-error cases with 40 replications.

V. Results

A. Error-free Link

Figure 3 shows the median, quartiles, and 10th/90th percentiles of the flow completion time (FCT) for 400 flows with the different congestion control algorithms and buffer sizes. The FCT is the time elapsed between sending the first request of a flow to the arrival of the first copy of the ACK for the last (50th) request. The median FCT for FASOR with the continuous flows and 2500 bytes buffer is 5.86% shorter than that of CoCoA and roughly the same as that of Default CoAP. With the 2500 bytes buffer, the congestion losses cause many RTOs. None of those retransmissions are unnecessary. The number of retransmissions per flow is shown in Figure 4 for the different cases. FASOR and Default CoAP perform the least number of retransmissions having roughly the same median for the number of retransmissions. A CoCoA sender computes a weak RTO for many retransmitted CoAP messages. Each weak RTO update increases the resulting RTO value. However, larger RTO values do not result in a decrease in the number of retransmissions in this case but the CoCoA senders surprisingly perform more retransmissions than the FASOR senders as more messages are dropped at the bottleneck. Retransmissions occur more frequently with CoCoA because the variable backoff factor mechanism uses the factor of 1.5 when the retransmission timer value is larger than 3 seconds. The larger number of retransmissions compresses the median, inter-quartile range and also the 10th percentile towards the 90th percentile implying slightly better fairness between the flows but consumes network resources and costs some energy in making those extra transmissions that will not arrive at the receiver. FASOR+token is able to acquire more RTT samples than FASOR lowering the RTO and, similarly to CoCoA, performs more retransmissions.

With 2500 bytes buffer, the FCTs for the random flows using CoCoA and FASOR are slightly longer than those of the continuous flows because starting each new short-lived client causes reinitialization of the congestion control related variables, resulting in slightly longer RTOs to retransmit the

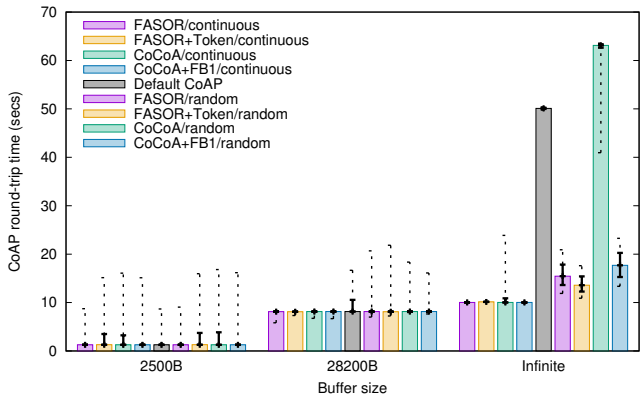

Fig. 5: CoAP round-trip time with 400 flows using different congestion control algorithms

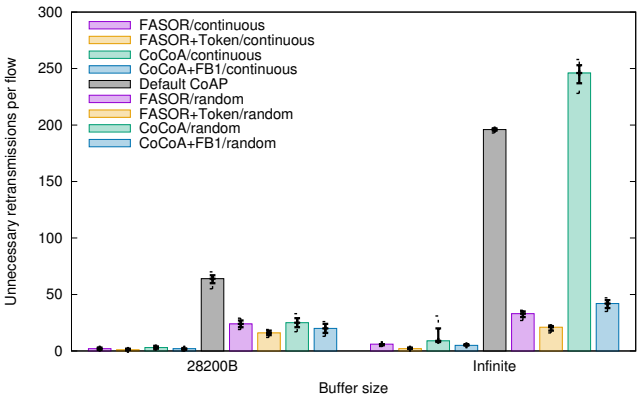

Fig. 6: Unnecessary retransmissions per flow (over 400 flows) using different congestion control algorithms

congestion losses. The median FCT for FASOR is 6.18% shorter than that of CoCoA.

In general, FCTs increase as the buffer size is increased because more CoAP messages fit into the buffer before the buffer overflows. As a result, less message drops occur due to congestion but the perceived CoAP RTT that is shown in Figure 5 increases. The CoAP RTT is the time elapsed from sending the original request to the first arriving ACK.

The median FCT for FASOR with continuous flows and 28200 bytes buffer is 2.81% shorter than that of CoCoA and 33.73% shorter than that of Default CoAP. With Default CoAP, the increase in buffer size causes the sender to perform a notably growing number of unnecessary retransmissions. Figure 6 shows the number of unnecessary retransmissions per flow with larger buffer sizes. With 28200 bytes buffer size, the median of unnecessary retransmissions per flow for Default CoAP is 64 messages. The reason why Default CoAP keeps making unnecessary retransmissions is twofold: Default CoAP sender does not base its RTO on measured RTT and it reverts the RTO back to the initial value (2-3 seconds) for every new CoAP exchange, that is, the sender resets the exponential backoff for every CoAP exchange starting again with the initial RTO that was already found too short during the previous exchange. Because of congestion, some of the unnecessary retransmissions are dropped at the bottleneck and only consume resources from the part of the end-to-end path before the bottleneck. The majority of the unnecessary retransmissions, however, make it past the bottleneck link wasting the bottleneck link capacity, and thereby, delaying the completion of the useful traffic.

For both FASOR and CoCoA, there is difference in the number of unnecessary retransmissions between continuous and random flows. As the prevailing RTT is more than the initial RTO, each newly starting short-lived client of a random flow performs some unnecessary retransmissions before it is able to get an ACK. The unnecessary retransmissions waste a portion of the link capacity causing the random flows to perform worse than the continuous flows for which the unnecessary retransmissions only occur at the beginning of 50 CoAP exchanges. FASOR+token performs the least number of unnecessary retransmissions because it is able to acquire an RTT sample already from the first exchange even if retransmissions occurred. Therefore, FASOR+token can adjust its RTO better than FASOR and, in general, with a larger weight than the CoCoA RTO calculation does. Less unnecessary retransmissions allow FASOR+token to achieve the shortest median FCT resulting in, for random flows, 2.63% and 4.91% shorter median FCT than with FASOR and CoCoA, respectively.

Figure 7 shows the FCT for 400 flows with the infinite buffer size that introduces a challenging bufferbloat network environment. Like with the 28200 bytes buffer, unnecessary retransmissions occur but with the infinite buffer size any unnecessary retransmission will always consume capacity of the bottleneck link away from useful traffic. The median FCT for FASOR with continuous flows is 14.07% shorter than that of CoCoA. CoCoA is able to control its RTO but still makes some unnecessary retransmissions at the beginning of the flow before it is able to adjust its RTO to the correct level. With continuous flows, the median of unnecessary retransmissions

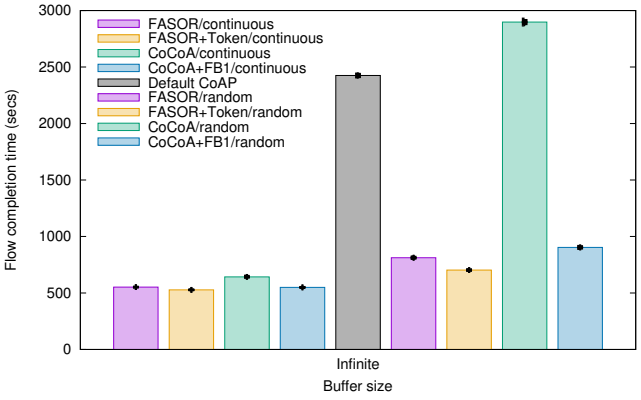

Fig. 7: Flow completion time for different congestion control algorithms with 400 flows and infinite buffer size

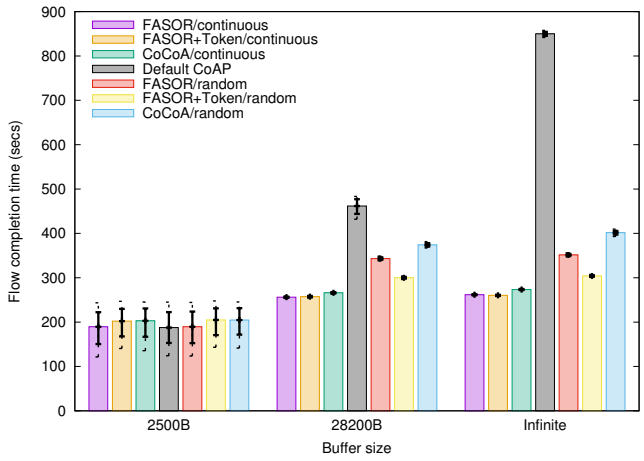

Fig. 8: Flow completion time with 200 flows using different congestion control algorithms

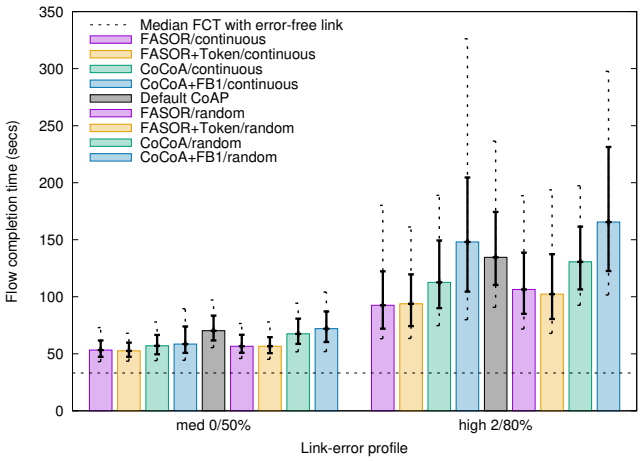

Fig. 9: Flow completion time with 10 flows using different congestion control algorithms and link-error profiles

per flow for CoCoA is 9 messages.

FASOR is second only to FASOR+token for which the median FCT is 4.39% and 13.46% shorter than for FASOR with continuous and random flows, respectively. With continuous flows, the median for unnecessary retransmissions per flow is 6 and 2 messages for FASOR and FASOR+token, respectively.

The median of unnecessary retransmissions per flow for Default CoAP is 196 messages with the infinite buffer size. It means that nearly 80% of the bottleneck link capacity is wasted with Default CoAP. This condition is known as congestion collapse where the increased traffic results in a decrease in the delivery of useful messages. With the infinite buffer, no messages are lost due to congestion but end-to-end delay increases dramatically due to excessive queuing delay that is bloated by the unnecessary retransmissions awaiting in the queue. As a result, the FCTs become very long, the median for FASOR being 77.25% shorter than that of Default CoAP.

The random flow is very demanding for any congestion control algorithm because only a very limited number of CoAP messages are exchanged by each short-lived client. Each newly starting short-lived client of a random flow makes some number of unnecessary retransmissions due to the use of the initial RTO before it is able to get the first ACK from the network but then FASOR rapidly proceeds to stabilize its RTO to a roughly correct value because of Slow RTO. With the random flows, CoCoA fails to sample the RTT when a new short-lived client starts. As a result, CoCoA uses its initial RTO for almost all CoAP message exchanges similar to Default CoAP making even more unnecessary retransmissions than Default CoAP because of the variable backoff factor [4]. Therefore, the median FCT for FASOR with random flows is 71.98% shorter than that of CoCoA. The median of unnecessary retransmissions per flow for FASOR with random flows is 33 messages which is 86.59% smaller than that of CoCoA.

When we apply the congestion safe variant of CoCoA (CoCoA+FB1) [4], the CoAP RTT, number of unnecessary retransmissions, and FCT with random flows decrease down to an acceptable level but remain at a somewhat higher level than those of FASOR as can be seen in Figures 5, 6, and 7.

Figure 8 shows the FCT for 200 flows with different congestion control algorithms and buffer sizes. With the 2500 bytes buffer, FCT trends are similar to those with 400 flows. With larger buffer sizes, the buffer is large enough to hold more than one request from each flow. Therefore, FCTs become extremely stable even though there are some unnecessary retransmissions except with Default CoAP that experiences a notable number of unnecessary retransmissions. With the 28200 bytes buffer size, FASOR achieves 3.73% and 8.27% shorter median FCT than CoCoA with continuous and random flows, respectively. With random flows, the median FCT for FASOR+token is 12.58% shorter than that of FASOR because of the improved RTT estimation. With the infinite buffer size, the trends are similar as with the 28200 bytes buffer.

B. Error-prone Link

Figure 9 shows FCTs for 10 flows with the error-free, medium and high link-error profiles. Congestion losses do not occur in this test because there are only 10 flows and the buffer size is 2500 bytes. Therefore, the median FCT with the error-free link is effectively almost the same with all algorithms. Link errors cause FCTs to increase compared to the results acquired with the error-free link because a sender needs to wait for an RTO to expire whenever a link-error related loss occurs. This waiting makes accurate RTT estimation very important.

With the medium link-error profile, the median FCT for FASOR with continuous flows is 6.58% and 24.15% shorter than that of CoCoA and Default CoAP, respectively. FASOR+token is able to lower the RTO faster than FASOR achieving 1.23% shorter median FCT than FASOR. With random flows, FASOR achieves 16.18% and 19.44% shorter median FCT than CoCoA and Default CoAP, respectively.

The Expired RTOs with different link-error profiles are shown in Figure 10 for the different congestion control algorithms. The expired RTOs is calculated over the values of

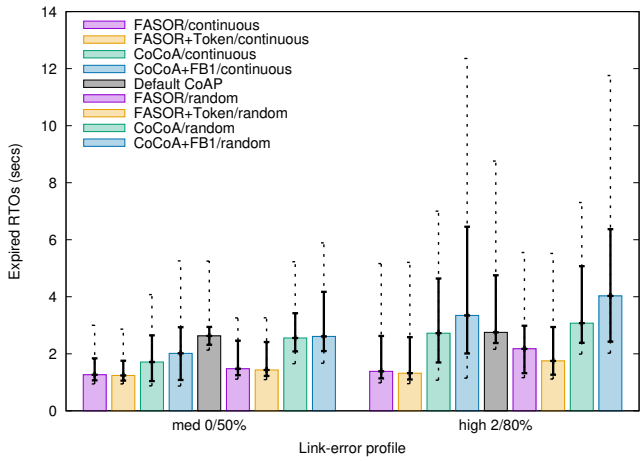

Fig. 10: Expired RTOs with 10 flows using different congestion control algorithms and link-error profiles

all RTOs that expired. With the medium link-error profile and continuous flows both CoCoA and FASOR manage to reduce the impact of the link errors because the RTO is armed often enough with a shorter value than with Default CoAP.

With the high link-error profile, the network conditions become challenging. The weak estimator in CoCoA collects many weak RTT samples for retransmitted messages that were dropped due to the link errors. This causes the median of expired RTOs for CoCoA to become longer than that of FASOR and with random flows even longer than that of Default CoAP. As a result, FASOR achieves 17.82% and 18.52% shorter median FCTs than CoCoA with the continuous and random flows, respectively. FASOR manages to restrain the median of the expired RTOs below that of Default CoAP although with the random flows there are only a few RTT samples before a new short-lived client starts forcing FASOR to start again from scratch. Armed with better RTT estimates, the median FCT of FASOR compared to that of Default CoAP is 31.25% and 22.34% shorter with the continuous and random flows, respectively. FASOR+token further shortens the median FCT by 3.86% with the random flows because it can take RTT samples also when the sender retransmitted to cover the losses due to the link errors.

It is worth noting that in these link-error cases both Default CoAP and CoCoA gain unfair advantage over FASOR from their too aggressive handling of retransmission timer backoff reset that, in turn, causes performance issues for them in the error-free cases with the infinite buffer size. With the congestion safe CoCoA+FB1 variant, the FCT clearly increases from that of CoCoA and spans over a larger range as can be seen in Figure 9. With the high link-error profile, FASOR compared to CoCoA+FB1 yields 37.5% and 35.69% shorter median FCT with continuous and random flows, respectively. The difference between CoCoA and CoCoA+FB1 shows what is the inherent cost of the congestion safety feature necessary for all traditional RTO based algorithms. FASOR, on the other hand, is able to handle not only the bufferbloat scenarios well but also manages to achieve shorter median FCTs than the congestion unsafe variants in the link-error cases.

VI. Conclusions and Future Work

This paper proposes FASOR, a novel RTO computing algorithm and backoff logic for congestion control, for CoAP. FASOR seeks to handle link-error related losses efficiently while handling congestion safely. The experiments in this paper show that FASOR achieves shorter flow completion times with link errors compared with Default CoAP and CoCoA and at the same time FASOR handles heavy congestion even in a bufferbloated environment well in contrast to Default CoAP and CoCoA that both fail to efficiently control congestion.

The current version of FASOR does not include special logic for senders remaining idle that is typical for CoAP but adding such logic might improve performance as more Slow RTOs could be avoided. Similarly, the current hard coded Slow RTO upper bound of 60 seconds may benefit from further improvements and evaluation whether 60 seconds is, in fact, long enough.

The benefits of FASOR seem obvious with typical CoAP transfers but also other protocols such as TCP may benefit from it. We leave testing FASOR for TCP as future work.

References

[1] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC 896, Jan. 1984.
[2] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014.
[3] C. Bormann, A. Betzler, C. Gomez, and I. Demirkol, "CoAP Simple Congestion Control/Advanced," Internet Draft, Feb. 2018, Work in progress.
[4] I. Järvinen, I. Raitahila, Z. Cao, and M. Kojo, "Is CoAP Congestion Safe?" in *Proc. Applied Networking Research Workshop 2018 (ANRW'18)*, Jul. 2018.
[5] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *ACM Queue*, vol. 9, no. 11, Nov. 2011.
[6] I. Järvinen, M. Kojo, I. Raitahila, and Z. Cao, "Fast-Slow Retransmission and Congestion Control Algorithm for CoAP," Internet Draft, Jun. 2018, Work in progress.
[7] 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.
[8] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298, Jun. 2011.
[9] C. Bormann and Z. Shelby, "Block-Wise Transfers in the Constrained Application Protocol (CoAP)," RFC 7959, Aug. 2016.
[10] A. Betzler, C. Gomez, I. Demirkol, and J. Paradells, "CoAP Congestion Control for the Internet of Things," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 154–160, Jul. 2016.
[11] ——, "CoCoA+: An Advanced Congestion Control Mechanism for CoAP," *Ad Hoc Networks*, vol. 33, pp. 126–139, 2015.
[12] F. Zheng, B. Fu, and Z. Cao, "CoAP Latency Evaluation," Internet Draft, Jul. 2016, Work in progress.
[13] 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.
[14] 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.
[15] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681, Sep. 2009.
[16] "libcoap: C-Implementation of CoAP." [Online]. Available: https://libcoap.net/