BDP-CoAP: Leveraging Bandwidth-Delay Product for Congestion Control in CoAP

Emilio Ancillotti, Raffaele Bruno
Institute of Informatics and Telematics (IIT)
National Research Council
Via G. Moruzzi, 1 - 56124 Pisa, ITALY
Email: {e.ancillotti,r.bruno}@iit.cnr.it

Abstract—**CoAP is one of the most popular protocols for data transfer in IoT networks. Since COAP uses an unreliable transport protocol (UDP) to deliver application data, loss-based congestion control algorithms are introduced in CoAP to mitigate network congestion. In particular, CoCoA+, which is currently under standardisation by the IETF, leverages RTT-measurements to regulate the frequency of packet retransmissions. Recent studies have shown that CoCoA+ still suffers from some critical performance issues, and a few modifications were proposed. In this paper, we follow a different approach, and we design a rate-based congestion control algorithm for COAP, called BDP-COAP, which is derived from the TCP BBR protocol. More precisely, BDP-COAP paces the transmissions of a CoAP sender in order to match the estimated bandwidth of the bottleneck link and constrains the total amount of unacknowledged data to be upper-bounded by the estimated bandwidth-delay product. We compare our solution against standard CoAP and CoCoA+. Results demonstrate the BDP-COAP significantly improves throughput fairness while obtaining similar total goodput as CoAP and CoCoA+. Furthermore, BDP-COAP ensures more stable performance also in dynamic traffic scenarios and when competing with congestion-unaware traffic.**

Index Terms—**Internet of Things, CoAP, congestion control, bandwidth-delay product, Cooja**

I. Introduction

IoT systems are usually built upon small, battery-powered devices, whose primary purpose is to collect data from the physical environment. Therefore, IoT devices typically have limited computation capabilities and rely on low-power communication technologies to exchange data [1]. Considering the limited available bandwidth and the dense deployments that are foreseen for typical IoT systems, it is essential to prevent network congestion in order to ensure the reliable and timely collection of sensed data [2].

Nowadays, one of the most popular protocols for data transfer in IoT systems is the Constrained Application Protocol (CoAP), which is standardised by the IETF Core Working Group [3]. In brief, CoAP is a lightweight version of HTTP, and it adopts the same request/response paradigm to allow applications to exchange data with IoT devices: the client makes a request, and the server issues a response that includes the requested content. Different from HTTP, which relies on TCP to transport application data, CoAP uses UDP. Since UDP does not support reliable communications, CoAP natively implements a transmission reliability mechanism, as well as a simple loss-based congestion avoidance scheme. A more sophisticated congestion control algorithm, called CoCoA [4], is also under standardisation, which leverages round-trip-time (RTT) measurements of data connections to adjust the retransmission rate in order to avoid frequent retransmissions that can lead to network congestion. Although it is shown that CoCoAP improves throughput and delay performance over CoAP, other studies have also highlighted some critical issues, such as excessive growth of RTO and vulnerability to congestion-unaware traffic [5]. Therefore, recent works have proposed various modifications to the CoCoA design to improve the RTO estimator and to better distinguish between wireless losses and congestion losses in order to reduce spurious retransmissions [6], [7].

The common design choice of CoAP, CoCoA, and their variants is to use packet losses as a signal of congestion. However, the design of loss-based congestion control algorithms is challenging in IoT systems as limited buffer sizes, lossy wireless channels, and link-layer contention makes packet losses very frequent. In this work, we leverage an alternative approach, initially proposed in the TCP BBR protocol [8], which employs estimates of the bottleneck bandwidth and round-trip propagation time to discover the optimal operating point of a data connection. More precisely, BBR paces the packet transmissions in order to match the estimated bandwidth of the bottleneck link for the data connection and constrains the total amount of unacknowledged data to be upper-bounded by the estimated bandwidth-delay product. One of the main contributions of this paper is the design and implementation of a new rate-based congestion control algorithm for CoAP, called BDP-CoAP, derived from the BBR protocol. The main novelty of BDP-CoAP is the redesign of the BBR's estimator of the bottleneck bandwidth to cope with lossy links and the short-term unfairness of channel access that is typically observed in IoT networks. We have carried out an extended performance evaluation comparing BDP-CoAP against standard CoAP and CoCoA+. Results demonstrated that BPD-CoAP significantly improves the fairness of data connections and reduce the number of retransmissions while achieving throughputs comparable to those observed with CoAP or CoCoA+. Furthermore, BDP-CoAP ensures stable performance also in dynamic scenarios or when competing with congestion-unaware traffic, while both CoAP and CoCoA+ suffer from instability and periods of starvation.

The rest of this paper is organised as follows. Background information about CoAP, CoCoA and BBR is provided in Section II. The proposed rate-based congestion control algorithm is described in Section III. In Section IV we present the results of

978-1-5386-4980-0/19/$31.00 ©2019 IEEE

the performance comparison. Finally, concluding remarks and future work are discussed in Section V.

II. Background

For the sake of clarity, this section provides an overview of the key features of the congestion control algorithms of CoAP and BBR.

A. CoAP

CoAP adopts a basic loss-based congestion control algorithm: packet losses are interpreted as a signal of congestion, and retransmissions are spaced out using a binary exponential backoff (BEB) policy. In standard CoAP, the initial value for the retransmission timeout (RTO) is randomly chosen from a fixed interval. Then, after every retransmission, the RTO is doubled. CoAP allows a total of four retransmissions of the same CoAP message before considering the transmission failed. Furthermore, CoAP limits the number of outstanding parallel transmissions towards the same destination to `NSTART`. By default, CoAP recommend to set `NSTART` to one (i.e., not more that one unacknowledged CoAP message per RTT to a client on average).

Alternative congestion control algorithms have been proposed for CoAP and are currently under standardisation within the activities of the CoRE Working Group. The most relevant of such proposal is the Simple Congestion Control/Advanced, also known as CoCoA [4]. As in conventional TCP, CoCoA exploits continuous measurements of the round-trip-time (RTT) between CoAP endpoints to limit the frequency of retransmitted packets. Specifically, CoCoA maintains two RTT estimators, one (the "strong estimator") for CoAP exchanges that complete without retransmissions, and one (the "weak estimator") for exchanges that require retransmissions, although only the first two retransmissions are considered. Based on these RTT measurements, strong and weak RTO values are calculated, using the same algorithm as TCP [9]. The overall RTO estimate is an exponentially weighted moving average computed from the strong and the weak RTO estimators. Then, the initial RTO value ($\text{RTO}_{overall}$) for the first transmission is randomly chosen from the interval $[\text{RTO}_{overall}, \text{RTO}_{overall} \times 1.5]$.

Different modifications and variants of CoCoA's congestion control algorithm have been proposed over time to improve its performance. The last revision of CoCoA is known as CoCoA+ [10]. The reader is referred to [10] for a complete illustration of the algorithm and the rationale behind it.

B. BBR

The bandwidth-delay product (BDP) of a data connection is defined as the product of its round-trip delay and the capacity of the bottleneck link (i.e., the slowest link in the path traversed by that connection). The bottleneck link is important for a TCP connection because it determines the connection's maximum data-delivery rate, and it is where persistent queue form. BRR is a rate-based congestion control protocol that tries to maintain the number of packets *in flight* (data sent but not yet acknowledged) equal to BDP [11]. To explain the rationale behind this choice, the authors is [11] identified three operational regions for RTT and delivery rate variation as a function of the number of packets in flight ($inflight$). The first region (called app-limited in [11]) is characterised by $inflight \leq BDP$. In this case, the connection does not use all the available bandwidth of the bottleneck link, and the RTT is equal to the round-trip propagation time (RT_{prop}). In the second region (called bandwidth-limited), the connection saturates the bottleneck link, and its throughout is upper-bounded by the bottleneck bandwidth ($BtlBw$), while the RTT increases since the queue at the bottleneck builds up. Finally, when $inflight \geq BDP + Q_{Btl}$, where Q_{Btl} is the capacity of the queue at the bottleneck link, the connection is buffer-limited, and packets can get lost due to buffer overflows. TCP's loss-based congestion control algorithm operates at the onset of the buffer-limited region, while BBR operates at the beginning of the bandwidth-limited region, where the connection runs with the highest throughput ($BtlBw$) and the lowest delay (RT_{prop}).

The optimal operating state for BBR is equivalent to require that the packet arrival rate at the bottleneck link equals $BtlBw$, and the total data in flight is equal to $BDP = (BtlBw \times RT_{prop})$. BRR estimates the round-trip propagation time as the minimum RTT observed over a time window W_R (which is typically tens of seconds to minutes). Instantaneous delivery rates can be simply computed as the ratio of data delivered to the time elapsed between two ACK receptions. Then, BBR estimates $BtlBw$ as the maximum observed delivery rate over a time window W_B (which is typically six to ten RTTs). In steady state, BBR paces transmissions at the $BtlBw$ estimate, while trying to have not more than BDP data in flight. In order to learn changes in the physical properties of the connection's path, BBR employs an approach called *gain cycling*. Specifically, the packet transmission rate is multiplied by a scaling factor called $pacing\_gain$. When $pacing\_gain > 1$, BBR transmit faster to check for $BtlBw$ increases. On the other hand, if $pacing\_gain < 1$, BBR sends slower to check for RT_{prop} decreases. Then, BBR alternates between probing phases when it tests for higher bandwidth ($pacing\_gain > 1$), probing phases when it tests for lower round-trip times ($pacing\_gain < 1$) and convergence phases when it uses the estimated fair share of bandwidth ($pacing\_gain = 1$). In the following section, we explain in detail how BDP-CoAP leverages the BRR's concept of cycling gain.

III. BDP-CoAP Design

BDP-CoAP replaces the loss-based congestion control algorithm of CoAP with a rate-based congestion control algorithm that leverages the same design rationale of BBR. Specifically, BDP-CoAP selects the delivery rate of CoAP messages equal to the estimated $BtlBw$ and keeps the data in flight bounded to the bandwidth-delay product. However, BDP-CoAP differs from BBR in two fundamental design decisions: (i) BBR estimates the bottleneck bandwidth as the windowed-max of instantaneous throughput measurements. However, short-term unfairness in channel access is typically observed in constrained IoT networks. Therefore, CoAP messages transmitted in a window may experience variable congestion conditions with nodes that capture the channel for short periods and obtain high instantaneous delivery rates. Such conditions would easily lead BBR to over-estimate the available bandwidth. BDP-CoAP uses an estimator that combines both maximum and minimum delivery rate measurements to derive $BtlBw$ estimates. (ii)

If a connection is app-limited, BBR's $BtlBw$ estimator does not consider the bandwidth samples provided by the in-flight packets. BDP-CoAP also discards bandwidth measurements from retransmitted CoAP messages as the current CoAP and CoCoA+ implementations cannot match a specific ACK with its CoAP message when retransmissions occur. It is important to remark that packet losses due to congestion or lossy links are likely in constrained IoT environments. Therefore, BDP-CoAP tracks the number of missed bandwidth samples over the observation period, and it leverages this information to make the $BtlBw$ estimator more or less aggressive. Furthermore, the BDP value is typically small in IoT environments due to low link data rates and small buffers. As a consequence, most of the time a CoAP sender is able to successfully transmits less than one packet per RTT, on average. Therefore, BDP-CoAP avoids using the first CoAP message after short-term connection starvations for estimating the $BtlBw$ parameter. Algorithm 1 presents a detailed pseudocode description of the BDP-CoAP operations when a CoAP message is sent, or an ACK is received.

When the data pacing timer expires BDP-CoAP invokes the `sendPacket` function. If the number of in-flight packets is greater than BDP it waits for an ACK or a retransmission timeout (line 3), otherwise it checks if there is a pending CoAP message to transmit (line 6). If there is not a packet waiting for transmission, and there are not in-flight packets the connection is assumed to be inactive or in a starvation condition, and the next packet to be transmitted is tagged as the first of a new burst (line 9). If there is a packet to transmit, BDP-CoAP checks if the connection is app-limited (lines 16 20). Then, BDP-CoAP updates the packet statistics that are needed for the $BtlBw$ and RT_{prop} estimators (lines 21 24), and it computes the $pacing\_gain$ (line 26) that is used to update the delivery rate of the connection (line 27). Similarly to BBR, DBP-CoAP cycles through a sequence of values for the $pacing\_gain$. However, there are two main differences between BBR and DBP-CoAP. First, retransmitted packets are not considered when updating the $pacing\_gain$ (line 2. Second, in BBR a cycle consists of eight phases, and each phase normally lasts for the estimated RT_{prop} time, while in BDP-CoAP a cycle consists of ten CoAP exchanges (i.e., it contains ten transmissions of new CoAP messages). In BDP-CoAP implementation $gain\_factor = 1.2$ and $leak\_factor = 0.8$.

When an ACK is received BDP-CoAP invokes the `receiveACK` function. First of all, RTT (line 2) and throughput (line 6) measurements are obtained to update the RT_{prop} (line 7) and $BtlBw$ (line 10) estimates, respectively. Formally, let T be the time when an ACK is received. The two estimators work as follow:

$$RT_{prop_T} = \min(rtt_t) \quad \forall t \in [T - W_R, T] \tag{1}$$

$$BtlBw_T = \alpha(f_r)\max(th_t) + [1-\alpha(f_r)]\min(th_t) \quad \forall t \in [T - W_B, T] \tag{2}$$

Similar to BBR, the RT_{prop} estimator at time T is the running min of RTT measurements over time window W_R (which is fixed and equal to 30 seconds in our implementation). The $BtlBw$ estimator is the weighted sum of the minimum and maximum throughput measurements over a time window W_B (which consists of the last ten transmission attempts, i.e., the last ten invocations of the `sendPacket` function). The weight $\alpha(f_r)$ depends on the frequency f_r of retransmitted packets over the time window W_B as expressed in equation (3). We remind that BDP-CoAP do no use throughput estimates from retransmitted CoAP messages. Thus, when there are many retransmitted packets (i.e., $f_r \geq 20\%$) the $BtlBw$ estimator gives higher value to the low throughput measurements. On the other hand, when the wireless channel is reliable (i.e., $f_r < 20\%$), very low throughput measurements would be likely

Algorithm 1 BDP-CoAP core functions

```
 1: function sendPacket
 2:     $BDP \leftarrow BtlBw \times RTprop$;
 3:     if ( $inflight \geq BDP$) then
 4:         return
 5:     end if
 6:     packet ← nextPacketToSend();
 7:     if ( $packet = \varnothing$ ) then
 8:         if ( $inflight = 0$ ) then
 9:             $is\_first\_packet \leftarrow$ true;
10:         else
11:             $app\_limited\_until \leftarrow inflight$;
12:         end if
13:         return
14:     end if
15:     $inflight \leftarrow inflight + packet.size$;
16:     if ( $app\_limited\_until > 0$) then
17:         packet.$app\_limited \leftarrow$ true;
18:     else
19:         packet.$app\_limited \leftarrow$ false;
20:     end if
21:     packet.$first \leftarrow is\_first\_packet$;
22:     packet.$send\_time \leftarrow$ now;
23:     packet.$delivered \leftarrow delivered$;
24:     packet.$delivered\_time \leftarrow delivered\_time$;
25:     sendToNetwork(packet)
26:     computePacingGain(packet.$rtx$)
27:     $next\_send\_time = mathrmpacket.size/(pacing\_gain \times BtlBw)$;
28:     timerCallbackAt(sendPacket, $next\_send\_time$)
29: end function
```

```
 1: function computePagingGain($rtx$)
 2:     if ( $rtx = 0$ ) then
 3:         $cycle\_index = (cycle\_index + 1) \bmod cycle\_size$
 4:         if ($cycle\_index = 0$) then
 5:             $pacing\_gain = gain\_factor$
 6:         else if ($cycle\_index = 1$) then
 7:             $pacing\_gain = leak\_factor$
 8:         else
 9:             $pacing\_gain = 1$
10:         end if
11:     end if
12: end function
```

```
 1: function receiveACK
 2:     $rtt \leftarrow$ now − packet.$send\_time$;
 3:     $inflight \leftarrow inflight$ − packet.$size$;
 4:     $delivered \leftarrow delivered$ + packet.$size$;
 5:     $delivered\_time \leftarrow$ now;
 6:     $th = (delivered$ − packet.$delivered)/$
            (now − packet.$delivered\_time$);
 7:     updateRTprop($rtt$, now)
 8:     if ( packet.$app\_limited$ = false && packet.$first$ = true ) then
 9:         if ( packet.$rtx = 0$ ) then
10:             updateBtlBw($th$, now);
11:         else
12:             updateMissedSamples(packet.$rtx$);
13:         end if
14:     end if
15:     if ( $app\_limited\_until > 0$ ) then
16:         $app\_limited\_until = app\_limited\_until$ − packet.$size$;
17:     end if
18:     if ( $is\_first\_packet$ = true ) then
19:         $is\_first\_packet \leftarrow$ false;
20:     end if
21: end function
```

due to short-term unfairness and they should be weighted less.

$$\alpha(f_r) = \begin{cases} 0.6 & \text{if } f_r < 20\% \\ 0.2 & \text{if } f_r \geq 20\% \end{cases} \quad (3)$$

Finally, it is important to point out that BDP-CoAP uses the pacing timer to manage both the first transmission of a CoAP message and its retransmissions. This means that the back-off mechanism that CoAP employs to set the retransmission timeout is disabled while the RTO timer is initialised as in CoCoA+ and kept constant during each CoAP request/response transaction.

IV. Performance Evaluation

In the following section, we compare the performance of BDP-CoAP with standard CoAP and CoCoA+ through simulations. To this aim, we use the Cooja platform, a simulation platform integrated into ContikiOS that allows to precisely emulate off-the-shelf wireless sensor node hardware, and to execute real code [12]. We implemented BDP-CoAP in ContikiOS 3.1. Cooja motes are used to emulate wireless sensor devices. The IEEE 802.15.4 communication technology is used in the simulations, while radio duty cycling (RDC) is deactivated. The ratio propagation model is the Multipath Ray-tracer radio Medium (MRM) model, which includes fading and multi-path effects [12], and we set its parameters to achieve a 100% success rate at ten meters and an interference range of about twenty meters.

We consider a network of 89 nodes deployed along concentric circles. The distance between circles is 10 meters. The network topology is build using the RPL routing protocol [13]. All nodes are CoAP servers that periodically send POST notifications to a data collector (a CoAP client) located at the center of the network. Each server can generate two types of traffic: $i)$ *background traffic*, which consists of non-confirmable CoAP messages, i.e., messages that do not need to be acknowledged, and $ii)$ *reliable traffic*, which consists of CoAP messages that request an ACK. Note that background traffic is congestion-unaware.

The metrics used in the performance evaluation are: $i)$ the *goodput*, measured as the total amount of data successfully received per unit of time; $ii)$ the *short-term fairness*, computed with the well-known Jain's Index over a time window of 30 seconds; $iii)$ the *retransmission index*, evaluated as the percentage of retransmitted packets over the total number of packet sent by the CoAP sources, and $iv)$ the *loss rate* at the application level.

Each simulation is replicated five times to compute average values with 95% confidence intervals.

A. Static Traffic Scenario

In the first set of experiments, we analyse the ability of BDP-CoAP to converge to a fair throughput share while fully utilising the available network capacity. To this aim, each CoAP server generates a new POST message as soon as an ACK is received (i.e., CoAP senders are saturated).

Figure 1(a) shows a boxplot of the Jain's Fairness Index sampled every 30 seconds. The results show that BDP-CoAP achieves a remarkable improvement in throughput fairness with respect to CoAP and CoCoA+. The average fairness

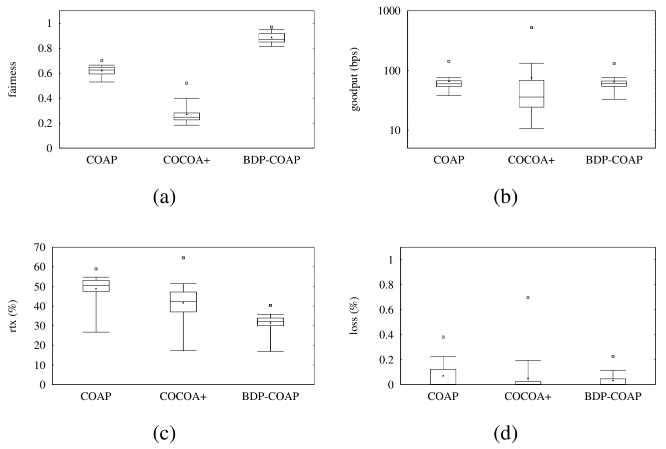

Fig. 1. Boxplots showing throughput fairness (a), goodput (b), retransmission index (c) and loss rate (d) for the static traffic scenario.

increases up to 470% and 194% compared to CoCoA+ and CoAP, respectively. This performance gain can be explained by considering the different policies used by each congestion control algorithm to manage packet retransmissions. Specifically, in multi-hop network topologies, the nodes that are closer to the data collector (or sink node) typically experience smaller retransmission indices than the farther nodes. When the offered load increases, the number or retransmissions also increases, and the use of a binary exponential backoff (BEB) mechanism to schedule retransmissions negatively affects the performance of the farthest nodes. This problem is further exacerbated in CoCoA+ because the RTO not only depends on the retransmission index but also on the RTT values that are estimated from retransmitted packets [5], [6]. By inspecting the RTO traces, we discovered differences between the average RTOs up to 90%.

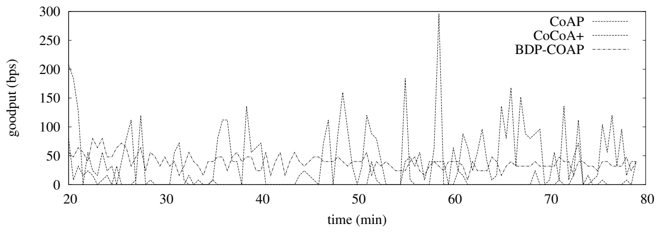

Fig. 2. Istantaneous goodput of a 4 hop node

Figure 1(b) shows a boxplot of the total goodput achieved by each CoAP sender. The results indicate that CoAP and DBP-CoAP perform similarly, while CoCoA+ experiences higher goodput variability, with few nodes that achieve high goodputs and other nodes that experience low performance. This can be explained by observing that the BEB algorithm can lead to an excessive growth of RTOs resulting in more unstable data connections. For the sake of example, Figure 2 shows the instantaneous goodput achieved by a node that is four hops far from the sink. While DBP-CoAP ensured a stable goodput, both CoAP and CoCoA+ exhibits a fluctuating behaviour, with long periods of starvation.

Figures 1(c) and 1(d) shows a boxplot of the retransmission index and the percentage of lost packets (i.e., CoAP messages that are discarded because retransmitted four times), respectively. The results demonstrate that the use of a pacing mech-

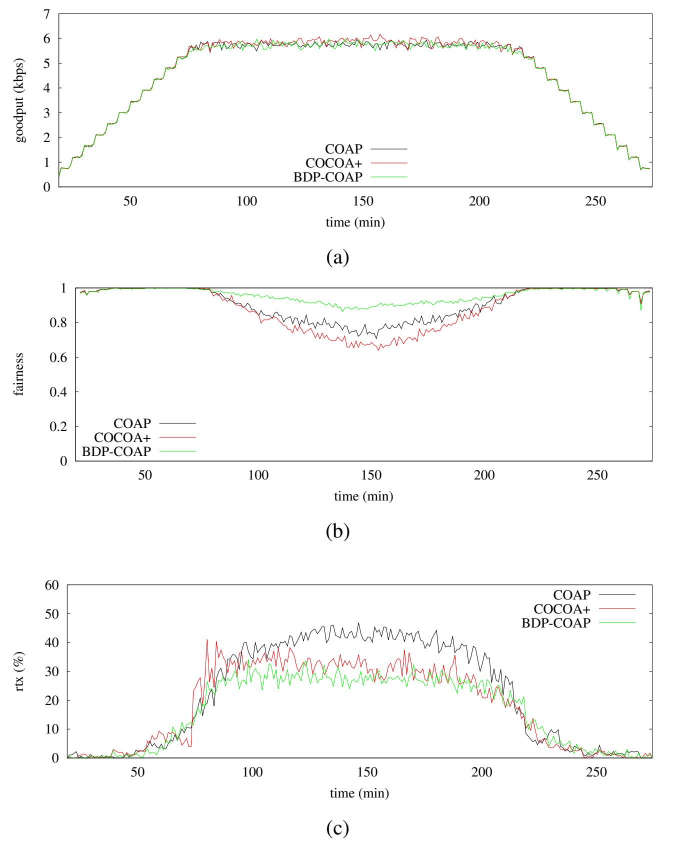

Fig. 3. Temporal evolution of goodput, fairness and retransmission index in the dynamic traffic scenario

anism for spreading out the packet transmissions reduces the network congestion, resulting in a lower retransmission index and packet loss. It is important to note that the use of an RTO independent from the RTT makes CoAP be more aggressive than CoCoA+, which explains why CoAP experiences a higher retransmission index and packet loss rates than CoCoA+.

B. Dynamic Traffic Scenario

In the second set of experiments, we investigate the ability of BDP-CoAP to adjust its sending rate to varying offered loads. To this aim, CoAP servers periodically send POST message to the sink and they increment the reporting frequency every five minutes with steps of 5bps from an initial rate of 8.5bps up to 136bps, and then back to 8.5bps with decrements of 5bps.

Figures 3 report the temporal evolution of the key performance metrics (aggregate goodput, throughput fairness, and retransmission index), sampled every 30 seconds. When the offered load of a CoAP sender is lower than 60bps, all the three mechanisms perform similarly and they are capable of properly satisfying the application requirements. However, when the offered load increases, the network contention also increases. For the nodes that are farther from the sink, the RTO estimate can become comparable with the application reporting period. Therefore, the CoAP senders that are closer to the sink start to capture an unfair share of the network capacity. The results not only demonstrate that BDP-CoAP behaves more efficiently than CoAP and CoCoA+ when the network congestion increases, but also that it converges quickly to the new optimal operating point.

C. Mixed Traffic Scenario

In the third set of experiments, we assess the impact of congestion-unaware traffic on CoAP flows. To this end, 50% of the nodes send confirmable POST message to the sink with a fixed rate equal to 680bps. The remaining nodes generate background traffic according to an on-off pattern. The duration of on and off period are the same and equal to six minutes. During the on period the node sends non-conformable POST messages with a fixed rate equal to 170 bps.

Figure 4 reports the goodput of individual CoAP senders as a function of time for all the nodes that generate congestion-aware traffic (i.e. confirmable POST messages). Several observations can be derived from the results. First, BDP-COAP ensures a stable goodput to every CoAP sender, while goodputs are highly variable with CoAP and CoCoA+. Second, there is a decrease during on period of background traffic, but all CoAP sources are able to deliver a non-negligible amount of packets. On the contrary many flows are starved with CoAP and CoCoA+, both during on and off periods of the background traffic.

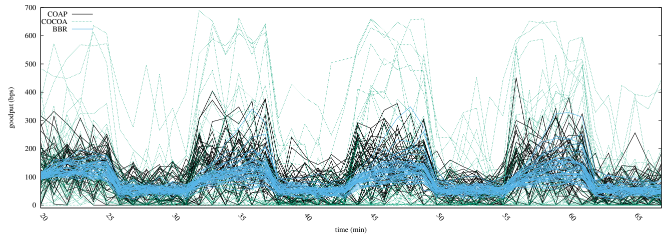

Fig. 4. Istantaneous Goodput of the worst nodes

V. Conclusions

In this paper, we presented BDP-COAP, a new rate-based congestion control algorithm for CoAP. The key idea of BDP-CoAP is inherited from the TCP BBR protocol: the bottleneck bandwidth and the round-trip propagation delays rather than packet losses or queuing delays are used to control the rate with which packets are retransmitted in the network. The design of an estimator of the bottleneck bandwidth in an IoT network is not a trivial task due to lossy channels, short-term unfairness for channel access, and low data rates.

We have compared DBP-CoAP against standard CoAP and CoCoA+ in a broad range of traffic scenarios. Results have demonstrated that BDP-CoAP is effective in ensuring throughput fairness and a low number of retransmissions while obtaining comparable total goodput as CoAP and CoCoA+. Moreover, differently from the other algorithms, BDP-CoAP provides stable performance also in dynamic traffic scenarios and with unresponsive background traffic.

Acknowledgment

This work has been partly funded by the European Commission through the FoF-RIA project AUTOWARE (No. 723909) and by the SIGS project, funded by the region of Tuscany under the FAR-FAS 2014 framework.

REFERENCES

[1] E. Borgia, “The Internet of Things vision: Key features, applications and open issues,” *Computer Communications*, vol. 54, pp. 1–31, 2014.

[2] C. Sergiou, P. Antoniou, and V. Vassiliou, “A Comprehensive Survey of Congestion Control Protocols in Wireless Sensor Networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1839–1859, Fourthquarter 2014.

[3] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” IETF RFC 7252, June 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7959.txt

[4] C. Bormann, A. Betzler, C. Gomez, and I. Demirkol, “CoAP Simple Congestion Control/Advanced,” Internet-Draft, Debruary 2018. [Online]. Available: https://tools.ietf.org/id/draft-bormann-core-cocoa-03.txt

[5] E. Ancillotti and R. Bruno, “Comparison of CoAP and CoCoA+ congestion control mechanisms for different IoT application scenarios,” in *Proc. of IEEE ISCC’17*. IEEE, 2017, pp. 1186–1192.

[6] S. Bolettieri, G. Tanganelli, C. Vallati, and E. Mingozzi, “pCoCoA: A precise congestion control algorithm for CoAP,” *Ad Hoc Networks*, vol. 80, pp. 116–129, 2018.

[7] R. Bhalerao, S. S. Subramanian, and J. Pasquale, “An analysis and improvement of congestion control in the CoAP Internet-of-Things protocol,” in *Proc. of IEEE CCNC’16*, January 2016, pp. 889–894.

[8] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “BBR: Congestion-Based Congestion Control,” *ACM Queue*, vol. 14, no. 5, pp. 50:20–50:53, October 2016.

[9] V. Paxson, M. Allman, J. Chu, and M. Sargent, “Computing TCP’s Retransmission Timer,” IETF RFC 6298, June 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6298.txt

[10] A. Betzler, C. Gomez, I. Demirkol, and J. Paradells, “CoCoA+: An advanced congestion control mechanism for CoAP,” *Ad Hoc Networks*, vol. 33, pp. 126–139, 2015.

[11] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “BBR: Congestion-Based Congestion Control,” *Queue*, vol. 14, no. 5, pp. 50:20–50:53, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/3012426.3022184

[12] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, “Cross-Level Sensor Network Simulation with COOJA,” in *Proc. of IEEE LCN’06*, November 2006, pp. 641–648.

[13] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur, “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks,” IETF RFC 6550, March 2012.