Multimodal Retransmission Timer for LPWAN

Carles Gomez, Jon Crowcroft

***Abstract*— Low Power Wide Area Networks (LPWANs) are experiencing high momentum as an inexpensive solution for enabling Internet of Things (IoT) applications. Recent Internet connectivity support developments are expected to further fuel the adoption of LPWAN. However, the latter present challenges to Internet protocols. Remarkably, many LPWAN scenarios exhibit a multimodal Round Trip Time (RTT) distribution, which deviates from common Internet RTT characteristics. This leads to a significant mismatch between Retransmission TimeOut (RTO), computed by the standard TCP or alternative experimental RTO algorithms, and RTT. In this paper, we present the Multimodal RTO algorithm, which is able to self-adapt to the current RTT mode and produce suitable RTO values. Evaluation results show that the Multimodal RTO reduces the RTO versus RTT misalignment of the TCP RTO algorithm by an average factor of up to 5, and reduces latency in the presence of losses by up to 2 orders of magnitude, while operating safely. The Multimodal RTO is currently being considered by the IETF as a candidate mechanism for standardization.**

***Index Terms*—Retransmission Timer, RTO, LPWAN, IoT.**

I. Introduction

In recent years, Low Power Wide Area Networks (LPWANs) have gained momentum as an inexpensive solution for enabling Internet of Things (IoT) applications. LPWAN technologies are based on star topology networks where IoT devices (e.g. battery-enabled sensors and actuators) are connected to a radio gateway via low-power, long-range wireless links, at the expense of reduced throughput [1-3].

As part of the expansion of LPWAN, IPv6 support is currently being developed for LPWAN [4, 5]. This capability will allow using IP-based protocols over LPWAN. Several such protocols use Retransmission TimeOut (RTO)-based packet retransmission for end-to-end reliability, and even the adaptation layer that enables IPv6 over LPWAN uses RTO-triggered retransmission [6]. However, some LPWAN technologies present unprecedented Round Trip Time (RTT) characteristics that challenge the performance of a traditional RTO algorithm such as the one used in the Transmission Control Protocol (TCP) [6, 7] or even of experimental, advanced RTO algorithms [8, 9].

In LPWANs, there exist two main phenomena that may exacerbate RTTs during some intervals. First, in some LPWAN technologies, as an IoT device energy saving technique, downlink messages can only be sent by a radio gateway to an IoT device during a specific time window shortly after an uplink transmission by the same IoT device [10, 11]. If the gateway is unable to send the response to an uplink message within the corresponding window, the next opportunity for the gateway to send the response will be after the next uplink transmission. The time at which the latter may occur depends on the specific application (e.g. the time between sensor readings are sent), potentially leading to an RTT several orders of magnitude greater than the one in ideal conditions [12]. Secondly, in some world regions, LPWAN technologies such as LoRaWAN and Sigfox conform to spectrum access regulations by keeping the duty cycle (DC) lower than a given limit (such as 1% in some bands in Europe) [11, 13]. Accordingly, some IoT devices remain in idle state after a packet transmission during 99 times the transmission time of the last packet sent. If a new packet needs to be sent during the idle interval, it will need to wait in a buffer of the IoT device until the next transmission is allowed, thus increasing the RTT.

Therefore, communication over LPWAN may happen under different sets of conditions, each one with different RTT characteristics, hereinafter referred to as *RTT modes*. In fact, many LPWAN scenarios exhibit multimodal RTT distributions, with a high RTT variance that stems from RTT mode changes. In order to provide high performance, an RTO algorithm for such RTT characteristics needs to be adaptive. However, the standard TCP RTO algorithm (hereafter, TCP RTO), which is the reference adaptive RTO mechanism used in the Internet, was not designed for multimodal RTT distributions. In fact, it underperforms in such scenarios, producing unnecessarily high RTO values after RTT mode transitions.

In this paper, we present and evaluate the Multimodal RTO algorithm (hereafter, Multimodal RTO). This algorithm comprises a number of internal timers that use the TCP RTO algorithm, each one intended for use in each RTT mode.

Based on LPWAN characteristics, we consider scenarios characterized by two RTT modes. As aforementioned, RTTs in LPWAN may be exacerbated due to compliance with duty cycle regulations or because gateways miss downlink transmission opportunities. However, in absence of such phenomena, RTTs are significantly lower. As a result, it is

This work has been funded in part by the Spanish Government (Ministerio de Ciencia, Innovacion y Universidades) through the Jose Castillejo grant CAS18/00170 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE.

C. Gomez is with the Network Engineering Department, Universitat Politècnica de Catalunya, Castelldefels 08860 Spain (e-mail: carlesgo@entel.upc.edu). His contribution to this work has been carried out during his research stay at the Computer Laboratory, University of Cambridge, Cambridge CB3 0FD United Kingdom.

J. Crowcroft is with the Computer Laboratory, University of Cambridge, Cambridge CB3 0FD United Kingdom (e-mail: jon.crowcroft@cl.cam.ac.uk).

possible to identify two main RTT modes, which we call High RTT mode and Low RTT mode, respectively. Note that, in some cases, at a microscopic scale, an RTT mode may comprise a number of RTT *submodes*. This may happen when an LPWAN device uses Automatic Repeat reQuest (ARQ) mechanisms at the link layer. In that case, link layer retries may lead to one specific submode for each specific number of retries. However, use of ARQ in LPWAN technologies is typically infrequent. For example, ARQ is not supported in Sigfox, whereas in LoRaWAN it is typically not recommended, since it would require using the downlink channel, which is a bottleneck, to send link layer acknowledgments [14].

The Multimodal RTO leverages a priori knowledge of RTT characteristics of an LPWAN, namely the number of RTT modes, and lowest and highest RTT values for each RTT mode. The Multimodal RTO adaptively adjusts its behavior according to the current RTT mode. By means of extensive simulations, we evaluate the TCP RTO, the Multimodal RTO, and other state-of-the-art RTO algorithms, in a range of LPWAN RTT scenarios. Results show that, in the scenarios considered, the Multimodal RTO outperforms the TCP RTO algorithm by reducing RTO and RTT mismatch by a factor up to 5, and by dramatically reducing packet delivery and confirmation latency in the presence of packet losses by up to 2 orders of magnitude, while offering safe operation. To our best knowledge, this is the first research work tackling the problem of RTO performance in LPWAN. On the other hand, the Multimodal RTO is currently being considered at the IETF as a candidate for standardization.

The main contributions of this paper are the following:

- Introducing and modeling the problem of multimodal RTTs in LPWAN.
- Presenting an RTO algorithm, called the Multimodal RTO, which, to the best of our knowledge, is the first RTO algorithm intended to address the multimodal characteristics of LPWAN RTTs.
- Evaluating the performance of the Multimodal RTO and comparing it with the performance of state-of-the-art RTO algorithms.

The remainder of the paper is organized as follows. Section II reviews related work in RTO algorithm design for TCP in general, and for IoT scenarios. Section III analyzes the High RTT mode in LPWAN. Section IV presents the Multimodal RTO. Section V evaluates the Multimodal RTO and the TCP RTO in a range of LPWAN RTT scenarios. Section VI expands the evaluation by comparing performance of the Multimodal RTO with that offered by relevant alternative state-of-the-art RTO algorithms. Section VII analyzes and discusses the complexity of the Multimodal RTO. Section VIII concludes this work.

II. Related Work

This section provides a literature overview in the area of RTO algorithms for the Internet. First, we focus on the TCP RTO, which can be considered the quintessential RTO algorithm used on the Internet. We provide the motivation for the design of the TCP RTO algorithm, along with a number of proposals intended to improve its performance. Secondly, we focus on RTO algorithms that have been specifically developed for the IoT domain.

A. Standard TCP RTO algorithm

The original specification of TCP, i.e. RFC 793, mandated use of a dynamically computed RTO [15]. While that specification did not require a particular RTO algorithm, it provided an example for such a dynamic RTO. The example was based on applying an Exponentially Weighted Moving Average (EWMA) to RTT samples in order to produce Smoothed RTT (SRTT) values. Then, the RTO was determined as a value proportional to the SRTT, with lower and upper bounds. However, Jacobson realized that lack of RTT variance estimation as an input to the RFC 793 RTO algorithm led to suboptimal performance [16]. He then proposed a modified RTO algorithm, with the aim of calculating RTO values based on both RTT mean and RTT variance estimates. Jacobson's algorithm was subsequently recommended for use in TCP as per RFC 1122 [17], and eventually became the standard TCP RTO algorithm in RFC 2988 [18]. As of the writing, Jacobson's RTO algorithm remains as the standard TCP RTO algorithm, albeit with a minor modification introduced in RFC 6298, which reduces the default initial RTO value (from 3 s to 1 s) [7]. Jacobson's algorithm, as specified in RFC 6298, operates as follows:

1) Until an RTT sample is obtained, *RTO* is set to 1 s.

2) When the first RTT sample is obtained, the Smoothed RTT (SRTT) is set to the value of that RTT sample (denoted *R*), RTT variation (RTTVAR) is set to *R*/2, and the RTO is calculated as

$$RTO = SRTT + \max(G, K \cdot RTTVAR), \quad (1)$$

where *G* is the clock granularity and *K* is equal to 4.

3) For any subsequent RTT sample, whose value is referred to as *R'*, RTTVAR and SRTT are updated as follows:

$$RTTVAR \leftarrow (1-\beta)\cdot RTTVAR + \beta\cdot|SRTT-R'| \quad (2)$$

$$SRTT \leftarrow (1-\alpha)\cdot SRTT + \alpha\cdot R', \quad (3)$$

where α=0.125 and β=0.25. Then, RTO is updated as per (1). If the computed RTO is smaller than 1 s, it is rounded up to 1 s, in order to keep TCP conservative.

An underlying assumption in Jacobson's RTO is a unimodal RTT distribution [19]. However, many LPWAN scenarios offer significantly different characteristics, such as a multimodal RTT distribution with very high variance.

B. Alternative RTO algorithms for TCP

An RTO algorithm represents a trade-off between latency and correctness, and thus no RTO algorithm will offer the best performance in all metrics and in all possible scenarios [20]. However, several works have identified TCP RTO calculation issues [21-25]. The impact of parameter settings, such as the initial RTO or the minimum RTO, on spurious timeouts has been investigated [22-24]. On the other hand, alternative RTO algorithms have been proposed as enhancements to the TCP RTO. Two prominent examples are Linux-RTO and Peak-Hopper RTO [25, 8].

Linux-RTO provides two additions to the TCP RTO. First, when a new RTT measurement is smaller than the SRTT, the RTO is not updated, which avoids an RTO increase when network conditions appear to yield lower RTTs. Secondly, Linux-RTO keeps a safety margin for RTO values when RTT samples offer constant values.

Peak-Hopper RTO computes short-term and long-term RTO values. When there is an unexpected RTT growth, Peak-Hopper RTO favors its short-term RTO value as the RTO algorithm output. Otherwise, the long-term RTO tends to prevail, in order to avoid spurious timeouts. A slow decay is applied to the long-term RTO.

As of the writing, none of the alternative TCP RTO proposals has achieved the same level of maturity and adoption as the standard TCP RTO.

C. IoT-specific RTO algorithms

RTO algorithm design for IoT environments has also attracted the attention of many researchers [9, 26, 27]. Work in this area has focused mainly on providing a suitable RTO algorithm for the Constrained Application Protocol (CoAP), a lightweight application-layer protocol designed and standardized by the IETF for constrained node networks [28]. CoAP was originally conceived to run over UDP, while supporting optional reliability (based on confirmations and RTO-based retries) at the application layer. The CoAP base specification establishes that the RTO is randomly chosen, by default, between 2 s and 3 s, regardless of network conditions. In order to improve CoAP performance, a proposal called CoCoA defines an adaptive RTO algorithm that uses RTT samples as input [9]. CoCoA can be considered the seminal adaptive RTO algorithm for CoAP.

CoCoA uses two internal TCP RTOs, one for strong RTTs (i.e. RTT samples that are obtained in absence of sender retries), and another one for weak RTTs (i.e. RTTs where the sender has performed retransmissions). The latter is intended to allow operation of CoCoA even in potentially lossy environments. The most recently updated internal RTO contributes to the overall estimator by using an EWMA algorithm, with default weights of 0.5 and 0.25 for the strong and the weak RTO estimators, respectively. In addition, CoCoA dithers the computed RTO value in order to avoid synchronization effects of neighboring devices such as sensors providing periodic updates. Other RTO algorithms have also been proposed for CoAP, such as pCoCoA [27] or FASOR [28]. However, they have been evaluated in a limited set of scenarios, in terms of network topologies and technologies.

Despite the existing body of RTO algorithm design in the literature, to the best of our knowledge, no previous RTO algorithm has been developed for multimodal RTTs.

III. High RTT mode in LPWAN: analysis

As introduced in Section I, RTTs in LPWAN may be exacerbated under a set of conditions, leading to a High RTT mode. This section analytically characterizes the High RTT mode in the two main scenarios where it arises: i) the *busy gateway* scenario, and ii) the *DC-constrained* scenario. For both scenarios, we derive the probability that an RTT belongs to the High RTT mode, denoted P_{High}, as well as the expected High RTT mode value, RTT_{High}.

A. Busy gateway scenario

In the busy gateway scenario, there exist time intervals during which the IoT device sends an uplink message, but the gateway misses the IoT device reception window for sending the corresponding downlink response. This happens when the gateway is busy performing other tasks, such as transmitting previously buffered downlink messages. As a result, the gateway typically sends the response to the uplink message after the next uplink message transmission.

Let us assume that the arrival of downlink messages at the gateway radio transmitter follows a Poisson model, with rate λ_{down}. Downlink messages comprise responses to uplink messages sent by IoT devices as well as unsolicited downlink messages. Let us assume a constant message transmission time, x, and an infinite gateway downlink buffer size. Under the mentioned assumptions, the gateway downlink transmitter can be modeled as an M/D/1 queuing system [29].

In the presented conditions, the IoT device reception window will be missed (i.e. an RTT will belong to the High RTT mode) if there exists at least one previous message in the gateway downlink transmitter system when the response to the corresponding uplink message arrives at the system. Then, P_{High} can be computed as follows:

$$P_{High} = \lambda_{down} \cdot x \quad (4)$$

In that case, the gateway is typically able to send the response to the (k-th) uplink message after the next ((k+1)-th) uplink message transmission. Since the uplink message interval is expected to be large in LPWAN (e.g. in the order of hundreds of seconds), and significantly greater than Low RTT mode values (e.g. in the order of seconds), the expected High RTT mode value, RTT_{High}, will be approximately equal to the expected uplink message interval.

B. DC-constrained scenario

In the DC-constrained scenario, an IoT device remains in idle state after sending a packet, in order to comply with a DC limit. For example, ETSI regulations establish a 1% DC constraint for the 868.0 – 868.6 MHz band in Europe [30], which is relevant for channels used by LoRaWAN and Sigfox. At the IoT device, any new message will need to wait in a buffer until all previous messages have been sent and their subsequent idle times have ended, therefore producing RTT values that belong to the High RTT mode.

We characterize message generation at the IoT device as a Poisson model, with rate λ_{up}. This may correspond to an application where an IoT device sends a message reporting a detected event, which may happen at any time, and independently from other events. Let us also assume a constant message transmission time, which is followed by a subsequent idle interval of constant duration (e.g. 99 times the last message transmission time), to comply with regulatory DC constraints. Let us define a service time, denoted τ, as the sum of uplink message transmission time and the subsequent idle interval. Note that τ has a deterministic value. Finally, let us assume a large device buffer size that can be modeled as an infinite queue (as in fact LPWAN messages may be very short, e.g. of just a few tens of bytes [3]).

Under the considered conditions, uplink message transmission in the DC-constrained scenario may be modeled as an M/D/1 queuing system. Accordingly, in such scenario, the probability of a message leading to a high RTT value, P_{High}, can be obtained as follows:

$$P_{High} = \lambda_{up} \cdot \tau \quad (5)$$

The expected High RTT mode value, RTT_{High}, can be computed as the expected waiting time in the queue (w_q) plus the Low RTT mode value, RTT_{Low}:

$$RTT_{High} = w_q + RTT_{Low} \quad (6)$$

Since w_q corresponds to waiting time in an M/D/1 queuing system, RTT_{High} can be expressed as shown next:

$$RTT_{High} = \frac{\lambda_{up} \cdot \tau^2}{2 \cdot (1 - \lambda_{up} \cdot \tau)} + RTT_{Low} \quad (7)$$

Assuming a typical DC constraint of 1%, and for $P_{High} > 10^{-1}$, which corresponds to uplink channel utilization greater than 10^{-3}, the queuing waiting time becomes dominant, and thus RTT_{High} can be approximated as shown next:

$$RTT_{High} \approx \frac{\lambda_{up} \cdot \tau^2}{2 \cdot (1 - \lambda_{up} \cdot \tau)} . \quad (8)$$

IV. Multimodal RTO algorithm

Motivated by the RTT characteristics of LPWAN, the Multimodal RTO is designed for scenarios with a multimodal RTT distribution, where RTT values can be categorized into one of at least two different RTT modes characterized by significantly different, mostly non-overlapping ranges of RTT values.

The Multimodal RTO comprises a number of separate internal retransmission timers, one for each RTT mode (Fig. 1, Algorithm 1). A separate TCP RTO algorithm is used for each internal retransmission timer. Note that, since the TCP RTO algorithm was designed for a unimodal RTT distribution, it is a good fit for the RTT characteristics of a particular RTT mode. Alternative RTO algorithms might be considered for the internal RTOs. However, they might not lead to significantly different performance in a multimodal RTT context, and would miss the advantage of using a well-known algorithm in terms of adoption, code reuse, etc. Based on measured RTT samples, the Multimodal RTO determines the current RTT mode, and selects the RTO values produced by the internal timer that corresponds to that RTT mode as the Multimodal RTO output.

As characterized in the previous section, RTT values can typically be categorized into two RTT modes in many LPWAN scenarios, namely a High RTT mode, and a Low RTT mode. Therefore, in this paper the Multimodal RTO uses two internal RTO timers.

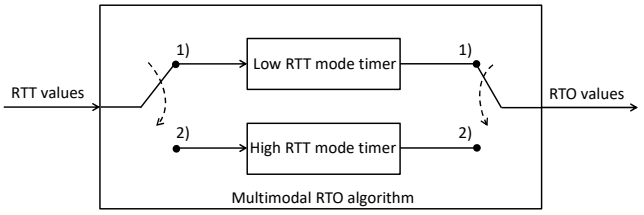

Fig. 1. Internal structure of the Multimodal RTO algorithm, which comprises two separate TCP retransmission timers intended for use in each RTT mode. When a new RTT mode is detected, the Multimodal RTO switches to using the corresponding internal timer (e.g. by switching from positions 1) to 2) when changing from Low RTT mode to the High RTT mode operation).

Algorithm 1

Input: *RTT vector of size N*[1]
Output: *RTO vector of size N*
Initialization:
  srtt_h ← 102
  rttvar_h ← 15
  current_mode ← LOW
  rtt ← 1
  rtt1 ← 1
  rtt2 ← 1
1: *srtt_l* ← *RTT[0]*
2: *rttvar_l* ← *RTT[0]/2*
3: *RTO[0]* ←*srtt_l* + max(*G*, *K*·*rttvar_l*)
4: **for** *i=1* to *i = N-1* **then**
5: *rtt2* ← *rtt1*
6: *rtt1* ← *rtt*
7: *rtt* ← *RTT[i]*
8: **if** *rtt* < *RTTthr* **then**
9: **if** *rtt1* < *RTTthr* **then**
10: **if** *rtt2* < *RTTthr* **then**
11: *current_mode* ← LOW
12: **end if**
13: **end if**
14: **else**
16: **if** *rtt1* ≥ *RTTthr* **then**
17: **if** *rtt2* ≥ *RTTthr* **then**
18: *current_mode* ← HIGH
19: **end if**
20: **else**
21: **if** *rtt2* < *RTTthr* **then**
22: *srtt_l* ← *RTT[i]*
23: *rttvar_l* ← *RTT[i]/2*
24: **end if**
25: **end if**
26: **end if**
27: **if** *current_mode* = LOW **then**
28: *srtt_l* ← (1-α)·*srtt_l* + α ·*rtt*
29: *rttvar_l* ← (1- β)·*rttvar_l* + β ·|*srtt_l*-*rtt*|
30: *RTO[i]* ←*srtt_l* + max(*G*, *K*·*rttvar_l*)
31: **else**
32: *srtt_h* ← (1-α)·*srtt_h* + α ·*rtt*
33: *rttvar_h* ← (1- β)·*rttvar_h* + β ·|*srtt_h*-*rtt*|
34: *RTO[i]* ← *srtt_h* + max(*G*, *K*·*rttvar_h*)
35: **end if**
36: **end for**

[1] *N* denotes the number of RTT samples.

The initial RTT mode assumed by the Multimodal RTO is the Low RTT mode. While in the Low RTT mode, if N_{High} consecutive RTT samples are equal to or greater than RTT_{Thresh_High}, the Multimodal RTO switches to High RTT mode operation. Analogously, if N_{Low} consecutive RTT samples are lower than RTT_{Thresh_Low} while in the High RTT mode, the algorithm switches to Low RTT mode operation.

An internal timer is frozen when the current RTT mode is not the one that the timer is intended for. Once a new RTT mode is detected, the corresponding internal timer reprises its

activity, using the last stored RTT mean and RTT variance estimates for this RTT mode. Therefore, in the Multimodal RTO, the relevant RTT variable estimates for a given RTT mode do not pollute the RTO calculation for another RTT mode. This behavior effectively avoids the persistently high RTO values produced by a single TCP RTO algorithm handling all RTTs, due to the high RTT variance produced by RTT mode changes.

We propose a setting of 3 for both N_{High} and N_{Low}, as a trade-off between reactivity to RTT mode changes and reliable RTT mode detection. Too large N_{High} or N_{Low} settings will delay internal timer change. On the other hand, too small N_{High} or N_{Low} settings may lead to spurious RTT mode detection. Both problems produce RTO values significantly greater than current RTT ones due to the resulting large RTT variance for the current internal timer in use. Optimal N_{High} and N_{Low} settings depend on the RTT features of each scenario.

As an additional measure to avoid underperformance due to sporadic high RTT samples while in Low RTT mode operation, if an RTT sample is greater than RTT_{Thresh_High}, and the next N_{Low} consecutive RTT samples are smaller than RTT_{Thresh_Low}, the Low RTT mode timer is reset.

V. Evaluation: Multimodal RTO vs TCP RTO

This section evaluates by simulation the performance of a single TCP RTO and the Multimodal RTO in a range of RTT scenarios. The section comprises five subsections. First, we describe the RTT models used in the evaluation. Secondly, we illustrate the behavior of the considered RTO algorithms in example scenarios. In the last three subsections, we compare the performance of the two RTO algorithms by focusing on three parameters: the average difference between the RTO values produced by the two RTO algorithms, the average RTO-to-RTT ratio for each RTO algorithm in each RTT mode, and the expected latency decrease achieved by the Multimodal RTO in the presence of packet losses.

A. RTT Models

In our evaluation, RTT values in the Low RTT mode are equal to 1 s, plus a uniformly random jitter of up to 0.1 s. Such Low RTT mode values are realistic in LoRaWAN networks [6]. For the High RTT mode, we consider the RTT patterns that arise in the busy gateway and DC-constrained scenarios.

In the busy gateway scenario, we assume a time between two consecutive uplink messages of 100 s. In these conditions, RTT values in the High RTT mode are modeled as 100 s, plus a random jitter of up to 10 s. Note that devices using LPWAN technologies will typically run applications that generate a relatively low rate of uplink messages. A time between two consecutive uplink messages of 100 s aims at capturing such application behavior, whereas the considered random jitter accounts for possible RTT variance. T_{Low} and T_{High} denote the average duration of intervals in the Low and High RTT modes, respectively. Let Ω denote T_{High}/T_{Low} ratio. We assume exponentially distributed RTT mode interval durations, expressed in number of RTT samples.

In the DC-constrained scenario, we assume a DC limit of 1%. We also assume a packet transmission time of 1 s, which is realistic for LPWAN technologies such as LoRaWAN (e.g. data rates called DR0 and DR1) and Sigfox (in Europe). In order to evaluate the RTO algorithms in adverse conditions (i.e. without limiting the RTT values), we assume a buffer of infinite size. In the study, the time between consecutively generated packets is exponentially distributed, with an average value of $1/\lambda$ s.

Note that the Multimodal RTO relies on a priori knowledge of the RTT distribution of the intended scenario. The main aspects and parameters to be determined are the number of RTT modes, suitable RTT mode thresholds, and suitable initial values for the internal timer variables. All these aspects can be determined a priori for both types of RTT patterns introduced in this subsection, as follows. First, RTT statistics need to be determined. This can be carried out by theoretical means, based on knowledge of the physical layer and link layer settings in use. Empirical RTT measurements can also be performed, either to refine the theoretical RTT characterization or if physical or link layer settings are unknown. The RTT characterization allows identifying the number of RTT modes. For a given RTT mode, initial values for the corresponding SRTT and RTTVAR variables will be suitable as long as they lead to a calculated RTO value that does not incur a spurious timeout, but is not too large either in order to avoid unnecessarily long wait time if retransmission is needed. A threshold between two adjacent RTT modes needs to be greater than the RTT values of the lower RTT mode, and smaller than the RTT values of the higher RTT mode.

In the evaluation, based on a priori knowledge of the scenario RTT characteristics, since RTT values smaller than 2 s are considered to belong to the Low RTT mode, and RTT values equal to or greater than 2 s correspond to the High RTT mode, both RTT_{Thresh_Low} and RTT_{Thresh_High} are set to 2 s. The goal is to avoid an internal Multimodal RTO timer use RTT samples not belonging to its intended mode. The SRTT and RTTVAR initial values for the Multimodal RTO in the High RTT mode are 102 s, and 15 s, respectively.

B. RTO Algorithms: Behavior Overview

We now provide an overview of how the TCP RTO and the Multimodal RTO outputs evolve over time for various example RTT sample sequences, in the busy gateway and DC-constrained scenarios (see Figs. 2 and 3, respectively).

Fig. 2.a) shows a sequence of RTTs that contains a High RTT mode interval with a duration of 100 RTT samples. Both considered RTO algorithms produce well adapted RTO values during the initial Low RTT samples. Once the RTT mode changes at the 100th RTT sample, high RTT variance leads to increased RTO values for both algorithms. However, the TCP RTO reaches roughly 3 times the current RTT in the High RTT mode, and it takes roughly 30 RTT samples for the TCP RTO to converge in the High RTT mode. In contrast, the Multimodal RTO detects the RTT mode change after the third consecutive RTT sample greater than RTT_{Thresh_High}, and switches to High RTT mode operation. Subsequently, the RTO values produced by the Multimodal RTO converge quickly, after roughly 10 RTT samples in the first High RTT

mode interval (only 3 RTT samples are required for subsequent RTT mode changes). Once the High RTT mode interval ends, both RTO algorithms initially produce increased RTO values due to the resulting RTT variance, similarly to their reaction to the first RTT mode change. In this transition, since the new RTT mode values are low, it takes even longer for the TCP RTO to converge. Instead, as soon as the Multimodal RTO detects the new RTT mode, it switches back quickly to Low RTT mode operation.

The overall performance improvement achieved by the Multimodal RTO depends on the durations of the different RTT mode intervals. Fig. 2.b) depicts an example that keeps Ω=1 as in the previous example (during the first 80 RTT samples), but for a reduced T_{Low} of 20 samples. For these settings, each RTT mode interval duration is shorter than the TCP RTO convergence time. However, the Multimodal RTO converges quickly when the RTT mode changes. Therefore, the relative improvement of the Multimodal RTO in this scenario is greater than in the previous one.

We now focus on the RTO algorithms' behavior in the DC-constrained scenario. In Fig. 3.a), $1/\lambda$ takes the shortest practical value for the considered conditions (i.e. $1/\lambda$=100 s). For most of the analyzed interval, both RTO algorithms offer the same performance, since most RTTs are greater than RTO_{Thresh_High} and RTO_{Thresh_Low} (i.e. 2 s), as a result of the high λ value considered. Fig. 3.b) illustrates an example for $1/\lambda$=1000 s, where high RTT mode intervals are short (e.g. of just one RTT sample). The TCP RTO converges slowly to low RTT values after the RTT spikes, whereas the Multimodal RTO is able to quickly detect the change to Low RTT mode values, producing RTO values close to the former.

The ability to determine the RTT mode allows the Multimodal RTO to produce RTO values that are lower than

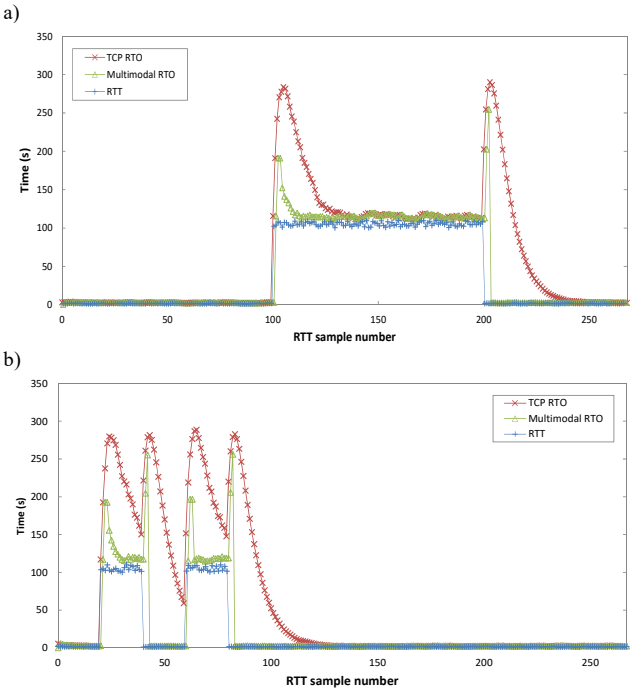

Fig. 2. Behavior of the TCP RTO and the Multimodal RTO in the busy gateway scenario: a) T_{Low} =100 samples, Ω=1; b) T_{Low} =20 samples, Ω=1 (during the first 80 samples).

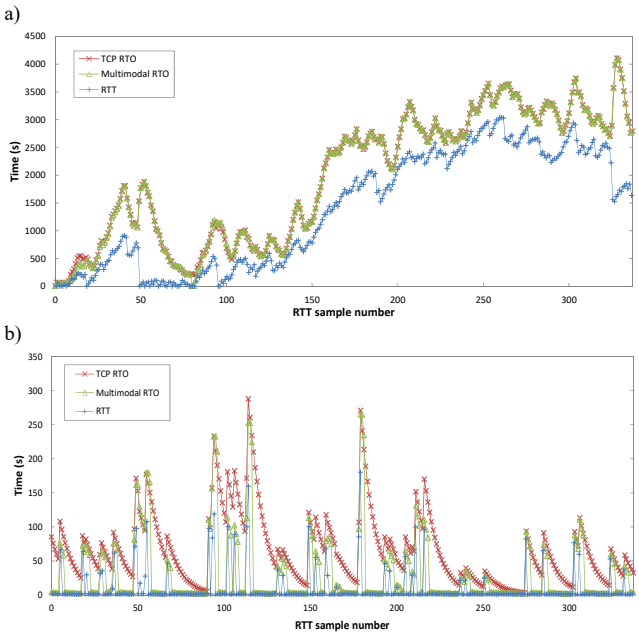

Fig. 3. Behavior of the TCP RTO and the Multimodal RTO in the DC-constrained scenario. a) $1/\lambda$=100 s; b) $1/\lambda$=1000 s.

the TCP RTO ones during transient intervals after RTT mode changes. On the other hand, the Multimodal RTO also provides safe operation, since its internal timers actually use the TCP RTO algorithm, thus the Multimodal RTO tends to produce RTO values that are greater than RTT values.

C. TCP RTO and Multimodal RTO output difference

We now study the average difference between the RTO values produced by the TCP RTO and the Multimodal RTO, denoted ΔRTO, in the busy gateway and in the DC-constrained scenarios. Each individual result shown in the remainder of this section is obtained from 10^6 simulated RTTs.

In the busy gateway scenario, we evaluate ΔRTO as a function of T_{Low} and Ω. Fig. 4 illustrates the corresponding results. The positive ΔRTO obtained in all cases indicates that the Multimodal RTO is able to follow the RTT more closely than the TCP RTO. For T_{Low}=10 samples or greater, ΔRTO tends to decrease with T_{Low}, since transient intervals after RTT mode changes become short compared with RTT mode interval durations.

All curves in Fig. 4 show a maximum ΔRTO. For low or high Ω values (e.g. 0.1 or 10, respectively), there is a dominant long RTT mode interval duration (in Low or High RTT modes, respectively) where both RTO algorithms converge to similar values, therefore ΔRTO is low. As T_{Low} increases, the maximum ΔRTO is found for a smaller Ω, because the High RTT mode interval duration needs to be smaller for transient intervals (where the difference between the two RTO algorithms is greatest) to remain dominant.

For low T_{Low} values (Fig. 4.b)), the *inverted U* shape of the curves becomes narrower as T_{Low} decreases. This happens because for low or high Ω (e.g. Ω=0.1 or Ω=10, respectively), the Multimodal RTO tends to behave like the TCP RTO, since the probability of Low or High RTT mode interval durations

being at least N_{Low} or N_{High} (i.e. 3) consecutive RTT samples decreases.

We now evaluate ΔRTO in the DC-constrained scenario. As shown in Fig. 5, ΔRTO exhibits a maximum for 1/λ = 200 s. As 1/λ increases, the duration of Low RTT mode intervals increases too, which allows the TCP RTO more time for converging during such intervals. For 1/λ= 100 s, both RTO algorithms offer similar performance, as e.g. previously shown in Fig. 3.a).

D. Per-mode RTO and RTT comparison

As it can be seen in Figs. 2 and 3, the relative performance of the TCP RTO and the Multimodal RTO varies significantly depending on the current RTT mode. The RTO and RTT mismatch of the RTO algorithms is significantly greater in the Low RTT mode. In this subsection, we evaluate the average RTO-to-RTT ratio (hereinafter, RTO/RTT) for the two studied RTO algorithms, separately for each RTT mode, for different T_{Low} and **Ω** values, in the busy gateway and DC-constrained scenarios (see Figs. 6 and 7, respectively).

As shown in Fig. 6.a), for T_{Low}=10 samples, during Low RTT mode intervals, RTO/RTT for the TCP RTO exceeds 100 for **Ω** ≥ 1, whereas the RTO/RTT is approximately 25 for the Multimodal RTO. Since duration of the Low RTT mode is short, the TCP RTO does not have enough time to converge. However, the Multimodal RTO converges quickly. For low **Ω** values, Low RTT mode intervals are long, and the impact of transient intervals becomes reduced.

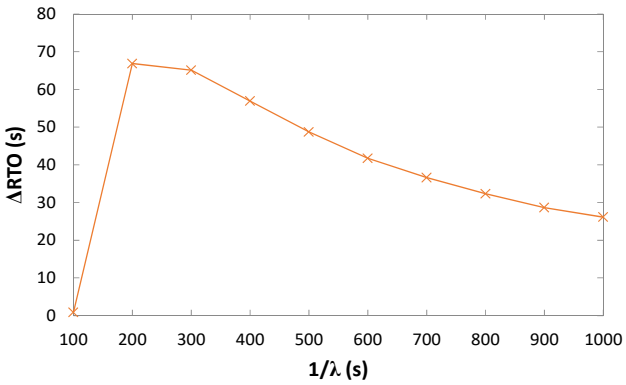

Fig. 5. ΔRTO in the DC-constrained scenario.

For T_{Low}=10 samples, but during High RTT mode intervals (Fig. 6.b)), RTO/RTT values are much lower. The TCP and Multimodal RTO algorithms exhibit local average RTO/RTT maxima of ~2.2 and ~1.7, respectively. For low **Ω**, the duration of the High RTT mode intervals is often short, which truncates the RTO values increase after a Low to High RTT mode transition, for both RTO algorithms. As **Ω** increases, probability of the mentioned truncation decreases, leading to an RTO/RTT increase. However, as **Ω** continues to grow, both RTO algorithms exhibit an asymptotic RTO/RTT decrease. Figs. 6.a) and 6.b) also provide the results for T_{Low}=100 samples, for the Low and the High RTT modes, respectively. The greater duration of each RTT mode interval for

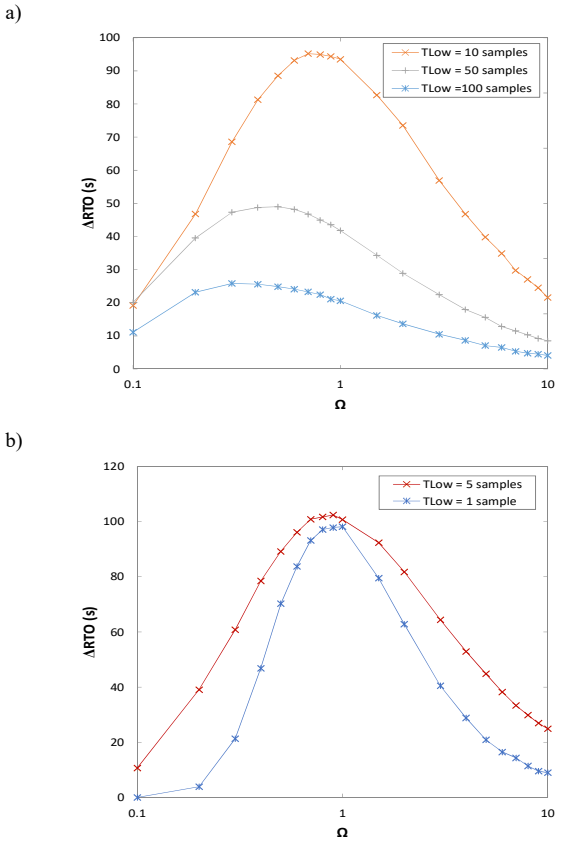

Fig. 4. ΔRTO in the busy gateway scenario, as a function of **Ω**: a) T_{Low} values of 10, 50 and 100 RTT samples; b) T_{Low} of 1 and 5 RTT samples.

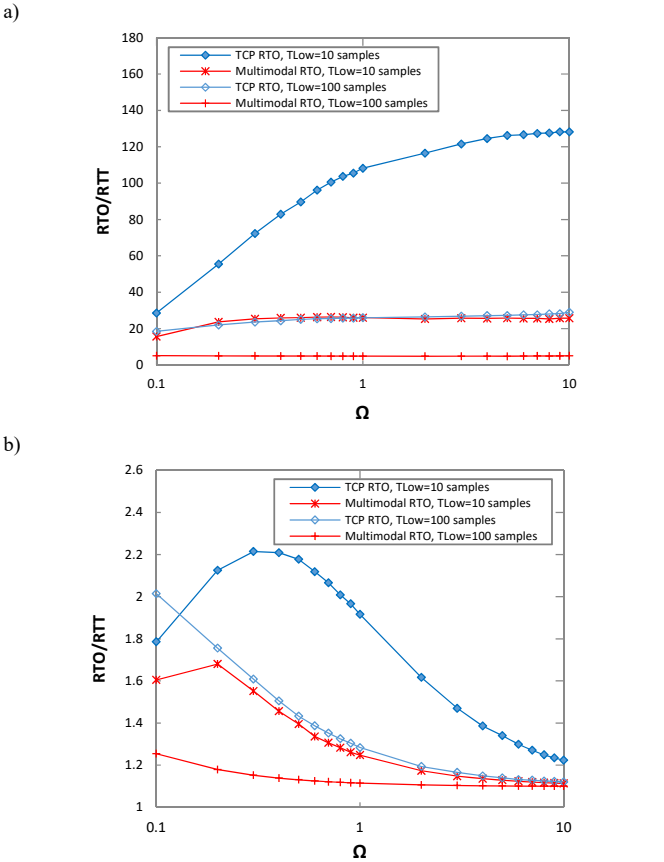

Fig. 6. Average RTO/RTT for the two considered RTO algorithms in the busy gateway scenario: a) Low RTT mode; b) High RTT mode.

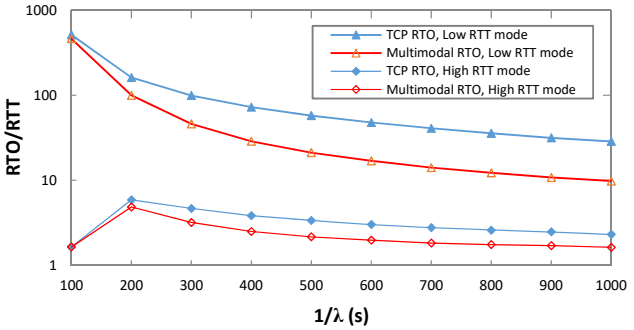

Fig. 7. Average RTO/RTT in the DC-constrained scenario, for the Low RTT mode and the High RTT mode.

T_{Low}=100 samples allows more time for the RTO algorithms to converge, leading to lower RTO/RTT in all cases.

Fig. 7 illustrates RTO/RTT for the analyzed RTO algorithms in the DC-constrained scenario. The greater RTT variance in this scenario leads to greater RTO/RTT than those obtained in the busy gateway scenario. In the Low RTT mode, RTO/RTT decreases with 1/λ, since Low RTT mode interval duration increases and the impact of transient intervals decreases. In the High RTT mode, there is a maximum RTO/RTT for 1/λ=200 s, similarly to the one shown in Fig. 5. The greatest difference between the RTO/RTT of the TCP RTO and the Multimodal RTO is given for 1/λ=300 s. Such difference also decreases with 1/λ, since impact of transient intervals decreases as well.

E. Latency Decrease

In the presence of packet losses, both considered RTO algorithms contribute latency to successful data packet delivery and confirmation (hereinafter, referred to as 'latency'). If a round trip is unsuccessful (i.e. either the data packet is not delivered or its confirmation is lost), the data packet is resent upon RTO expiration, and the next RTO value for the packet is duplicated. Note that this behavior is intended to keep the same conservative approach for the sender as in TCP, which is the benchmark for congestion control in the Internet, and ensures safe network operation. Since the Multimodal RTO generates RTO values that are in average lower than those of the TCP RTO, the Multimodal RTO yields a latency decrease. In this subsection, we evaluate such latency decrease in the presence of packet losses, in the same RTT scenarios considered in the previous two subsections, and for different values of the probability of unsuccessful round trip, denoted *P*. Note that, while in TCP there is a maximum RTO of 60 s [7], we assume that there is no RTO limit for any of the studied RTO algorithms, since time scales in LPWAN are much greater than the typical ones on the Internet [6]. We also assume that the number of retries is not limited, in order to determine the upper bound of the Multimodal RTO improvement. Figs. 8 and 9 depict the results, for the busy gateway and DC-constrained scenarios, respectively.

As shown in Figs. 8 and 9, the Multimodal RTO may achieve significant latency decrease values. For near-zero *P*, results shown in Figs. 8 and 9 are similar to those depicted in Figs 4 and 5. In the busy gateway scenario, in the presence of losses, average latency decreases by up to ~136 s and ~47 s, for T_{Low} values of 10 and 100, respectively, with a peak latency decrease of up to 2 orders of magnitude. In the DC-constrained scenario, packet latency decreases by up to ~95 s. Greater values of *P* amplify packet latency decrease.

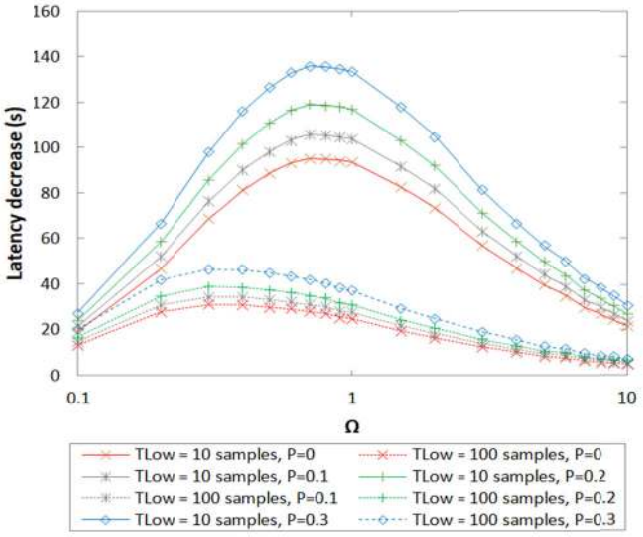

Fig. 8. Latency decrease achieved by the Multimodal RTO when compared with the TCP RTO in the busy gateway scenario, for T_{Low}=10 samples and T_{Low}=100 samples.

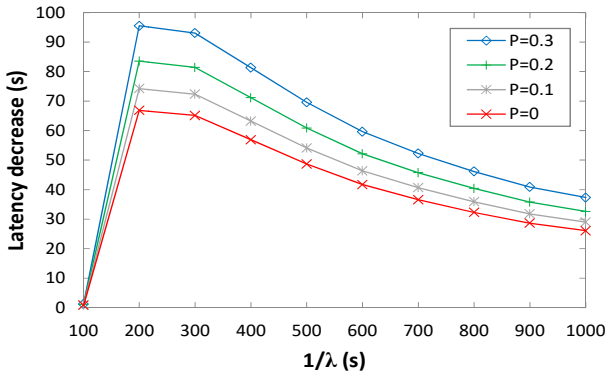

Fig. 9. Latency decrease achieved by the Multimodal RTO when compared with the TCP RTO in the DC-constrained scenario.

VI. Evaluation: Multimodal RTO vs Alternative RTO Algorithms

In the previous section, we evaluated the Multimodal RTO and the TCP RTO. In this section, we compare the performance of the Multimodal RTO and two relevant alternative, state-of-the-art RTO algorithms, namely: the Peak-Hopper RTO, and the CoCoA RTO (see Section II). Regarding the CoCoA RTO, in order to avoid an advantage for the rest of RTO algorithms in the comparison, in the evaluation we assume a CoCoA RTO version that does not use the weak estimator or dithering (since both tend to increase the computed RTO [9]).

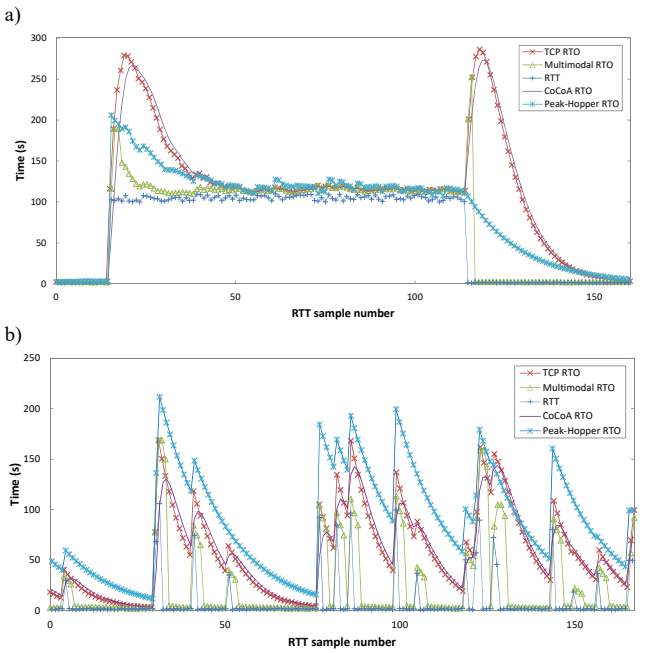

Fig. 10. Behavior of the Peak-Hopper RTO, the CoCoA RTO, the TCP RTO and the Multimodal RTO: a) busy gateway scenario, for T_{Low} =100 samples and Ω=1; b) DC-constrained scenario, for 1/λ=1000 s.

The section comprises two subsections. The first one overviews the behavior of both Peak-Hopper and CoCoA RTO algorithms, compared with that of the Multimodal RTO. Since the CoCoA RTO behaves similarly to the TCP RTO, the second subsection evaluates the Multimodal RTO, in terms of latency decrease, when compared with the Peak-Hopper RTO.

A. Behavior overview

Fig. 10.a) illustrates the RTO values produced by the Peak-Hopper RTO, the CoCoA RTO, and the Multimodal RTO in the busy gateway scenario, for a sequence of RTTs that contains a High RTT mode interval with a duration of 100 RTT samples. TCP RTO values are also included for comparison.

As it can be seen in Fig. 10.a), the CoCoA RTO behaves very similarly to the TCP RTO. In fact, the CoCoA strong estimator uses a TCP RTO, before an EWMA operation is applied. The EWMA scheme acts as a low-pass filter that slightly reduces the peak RTO value, delays the RTO adaptation to RTT mode changes by around 8 samples, and subsequently produces RTO values that are slightly greater than those of the TCP RTO.

In the busy gateway scenario, the Peak-Hopper RTO outperforms both the TCP and the CoCoA RTOs. While the Peak-Hopper RTO is able to react to the sudden RTT increase at the start of the High RTT mode interval, its RTO output remains lower than that of the TCP RTO and the CoCoA RTO. The latter are affected by a large RTT variance due to the RTT mode change. After the RTT mode change, the Peak-Hopper RTO decays its RTO value, and for ~20 RTT samples it produces RTO values much closer to the RTTs than those produced by the TCP and the CoCoA RTOs. Nevertheless, the Multimodal RTO outperforms the Peak-Hopper RTO.

When the High RTT mode interval ends, the Peak-Hopper RTO yields an even better performance (compared to that of the TCP and the CoCoA RTOs) than at the High RTT mode interval start. This occurs because as RTT decreases, the Peak-Hopper RTO decays its RTO output, whereas the TCP and the CoCoA RTOs produce large RTO values due to the high RTT variance due to the RTT mode change. At the third RTT sample after the RTT mode change, the Multimodal RTO detects the RTT mode change and switches to using its Low RTT mode internal timer. Subsequently, the Multimodal RTO produces RTO values well aligned with the current RTT mode. However, the Peak-Hopper RTO produces greater RTO values for at least 50 RTT samples.

The CoCoA RTO inherits many features of the TCP RTO, including its design for a unimodal RTT distribution. Thus, both algorithms significantly underperform the Multimodal RTO in the context of LPWAN scenarios. While the Peak-Hopper RTO outperforms both the TCP and the CoCoA RTOs in the busy gateway scenario, its design for a unimodal RTT distribution still prevents it from offering good performance in a multimodal RTT scenario.

Fig. 10.b) shows the RTO outputs for the Peak-Hopper, CoCoA, the TCP and the Multimodal RTOs, for an example sequence of RTTs in the DC-constrained scenario, for 1/λ=1000 s. As is visible in Fig. 10.b), the Peak-Hopper RTO offers the worst performance among the considered RTO algorithms. In the DC-constrained scenario, RTTs abruptly increase by at least one order of magnitude, and then return to lower values. The Peak-Hopper RTO reacts quickly to a sudden RTT increase by producing an RTO value that is greater than the current RTT, since its short RTT history becomes dominant. Once the RTT returns to low values, the long-term RTO of the Peak-Hopper algorithm prevails, which is however strongly influenced by the large RTO produced after the last RTT spike. The long-term RTO slowly decays over time, until the next high RTT sample, where the Peak-Hopper RTO value increases again.

In the DC-constrained scenario, the TCP RTO becomes less misaligned with the RTT sequence than the Peak-Hopper RTO, since the former does not neglect the longer past history of RTT samples, and short High RTT mode durations (of e.g. 1 RTT sample) do not suffice to produce higher RTO values than Peak-Hopper RTO ones. When the RTT sequence returns to low values after RTT spikes, the TCP RTO produces lower RTO values as well, avoiding the RTO exacerbated increase exhibited in the busy gateway scenario after a High to Low RTT mode change.

In the same scenario, the CoCoA RTO behaves similarly to the TCP RTO algorithm, with the CoCoA RTO leading to smoother RTO peaks, due to the low-pass filter effect of its EWMA component. In contrast with the rest of RTO algorithms considered, the Multimodal RTO is able to quickly adapt to the current RTT mode.

B. Latency decrease

We now study the latency decrease achieved by the Multimodal RTO in the presence of packet losses, compared with the Peak-Hopper RTO, for different values of *P*. As in subsection V.E, we assume unlimited RTO values and number of retries. Figs. 13 and 14 depict the results for the busy gateway and DC-constrained scenarios, respectively, as a

function of T_{Low} and Ω. Each individual result is obtained from 10^6 simulated RTTs.

As indicated by the positive latency decrease values in Fig. 11, the Multimodal RTO produces RTO values that are closer to RTTs than the Peak-Hopper RTO. However, the amount of improvement is lower (e.g. 45% lower for P=0) than when the Multimodal RTO is compared with the TCP RTO (Fig. 8). The reason is the lower misalignment of the Peak-Hopper RTO values with RTTs, compared to the behavior of the TCP RTO in the busy gateway scenario. The improvement tends to decrease with T_{Low}, since transient intervals after RTT mode changes become less relevant compared with RTT mode interval durations.

Similarly to Fig. 8, latency decrease curves in Fig. 11 also correspond to the general shape of an inverted U. However, the maximum values of the curves correspond to lower Ω values, in comparison with the ones shown in Fig. 8. Also, when T_{Low} increases, the maximum latency decrease is found for a smaller Ω in Fig. 11. Both phenomena occur because the High RTT mode interval duration needs to be smaller for transient intervals to remain relevant when comparing the Peak-Hopper and the Multimodal RTOs.

In the DC-constrained scenario (Fig. 12), latency decrease is greater than the one shown in Fig. 9 for all considered values of 1/λ. As explained in the previous subsection, the

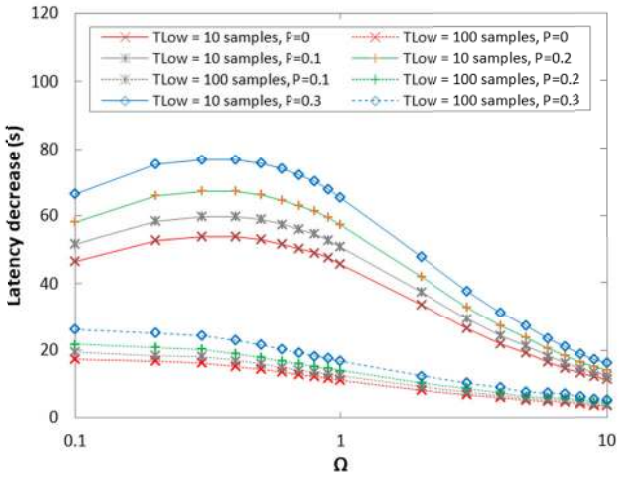

Fig. 11. Latency decrease achieved by the Multimodal RTO, compared with the Peak-Hopper RTO, in the busy gateway scenario, for T_{Low}=10 samples and T_{Low}=100 samples.

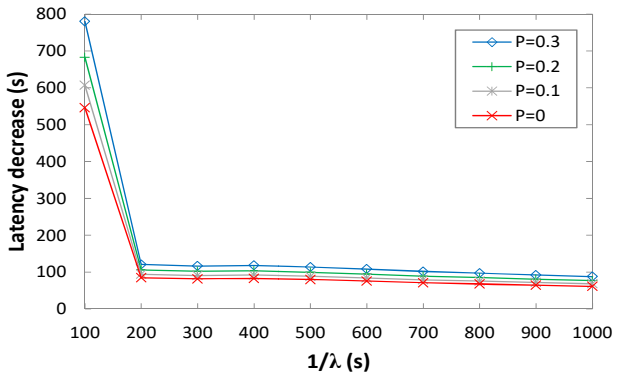

Fig. 12. Latency decrease achieved by the Multimodal RTO, compared with the Peak-Hopper RTO, in the DC-constrained scenario.

Peak-Hopper RTO underperforms the TCP RTO in the DC-constrained scenario. In addition, when RTT increase occurs for several consecutive RTT samples, the RTO output produced by the Peak-Hopper RTO is exacerbated. As 1/λ increases, high RTT values are less frequent, thus latency improvement decreases.

VII. Complexity Considerations

This section focuses on the complexity of the Multimodal RTO. First, a complexity analysis of the Multimodal RTO is provided; as a benchmark, complexity of the TCP RTO is also analyzed. Secondly, a discussion on the criteria for selecting the number of internal RTO timers of the Multimodal RTO, and the related trade-offs, is given.

A. Complexity analysis

We first evaluate time complexity, i.e., the time needed to produce an RTO output value on the basis of an RTT sample input. Secondly, we analyze spatial complexity, that is, the amount of memory required by an RTO algorithm to produce its output value. For the Multimodal RTO, we assume the same settings used in previous sections.

1) Time complexity

The TCP RTO requires obtaining SRTT and RTTVAR values in order to compute the RTO value. For computing SRTT, two multiplication and one addition operations are needed. For RTTVAR, two multiplications, one addition and one substraction are required. Once SRTT and RTTVAR values are available, one further multiplication and one addition are needed in order to compute the RTO. Let m, a and s denote the time required for performing a multiplication, an addition and a substraction, respectively. The time complexity for the TCP RTO is thus $5m + 3a + s$.

In the Multimodal RTO, the first step is determining the current RTT mode, and thus the internal RTO to be used. Let the value for either RTT_{Thresh_Low} and RTT_{Thresh_High} (which are assumed to be equal) be denoted RTT_{Thresh}. The current RTT sample value needs to be compared with RTT_{Thresh}. If no RTT mode change is detected, then the algorithm proceeds to computing the RTO output. Otherwise, whether up to the previous 2 RTT sample values correspond to a different RTT mode may need to be checked (note that we assume that N_{Low} and N_{High} are equal to 3). Let c denote the time required for performing a comparison operation. In the worst case, the time complexity of determining the current RTT mode is $3c$. Once the internal RTO to be used is determined, time complexity of computing the output RTO value is the same as that of the standard TCP algorithm, since each internal RTO of the Multimodal algorithm uses the TCP RTO. Therefore, the total time complexity for the Multimodal RTO is $3c + 5m + 3a + s$. Thus, the Multimodal RTO adds the time complexity of determining the current RTT mode.

2) Spatial complexity

To produce an RTO value, the TCP RTO stores the values of three variables: current RTT sample, SRTT and RTTVAR. Assuming that each variable requires a memory space of n, the spatial complexity of the TCP RTO is $3n$.

For the Multimodal RTO, the current sample, and 2 previous RTT samples need to be stored. In addition, for

each one of the two internal RTOs, SRTT and RTTVAR need to be maintained as well. Therefore, for the Multimodal RTO with default settings, spatial complexity is $3n + 2 \cdot 2n$, that is, $7n$. Thus, spatial complexity of the Multimodal RTO grows linearly with the number of internal RTO modes, and also includes the memory needed to store the last 3 RTT samples.

B. Selecting the number of internal RTO timers of the Multimodal RTO

As illustrated in the previous subsection, both spatial and time complexity of the Multimodal RTO increase with the number of internal RTO timers used.

In terms of accurate RTO computation, the number of internal RTO timers of the Multimodal RTO is optimally selected when it equals the number of RTT modes in the scenario. As explained in Section I, many common LPWAN scenarios are characterized by exactly two RTT modes, thus using two internal RTO timers for the Multimodal RTO is the best choice in such scenarios.

Many LPWAN devices exhibit limited hardware (e.g. memory, CPU, etc.) capabilities, thus computational power of such devices is constrained, and minimizing complexity is an important goal. If there are more than two RTT modes in a scenario, a reasonable trade-off between accurate RTO computation and minimizing complexity is defining two internal RTO timers for the Multimodal RTO, with one of them intended to cope with the High RTT mode (produced by duty cycle or busy gateway effects), and the other intended for the Low RTT mode (which may include submodes produced by link layer retries). In such case, while using only two internal RTO timers may not allow achieving the highest performance, it will still improve performance compared with a unimodal RTO.

VIII. Conclusion

In this paper, we presented the Multimodal RTO algorithm, which has been designed in order to address the issues posed by multimodal RTT distributions found in LPWAN scenarios. Evaluation results show how, at the expense of a relatively low amount of additional complexity, the Multimodal RTO outperforms use of a single, standard TCP RTO algorithm, as well as state-of-the-art RTO algorithms such as the Peak-Hopper RTO and the CoCoA RTO.

The TCP RTO fails to handle the large RTT variance that stems from RTT mode changes. The performance improvement that can be achieved by the Multimodal RTO over the TCP RTO increases for a moderate RTT mode change frequency and for relatively similar RTT mode interval durations of the different RTT modes. The CoCoA RTO performs similarly to the TCP RTO, and exhibits comparable misbehavior.

The Peak-Hopper RTO offers lower misalignment than TCP RTO or CoCoA RTO with RTTs in the busy gateway scenario. However, it fails to adapt quickly to an RTT mode change. Furthermore, in the DC-constrained scenario, the Peak-Hopper RTO underperforms even the TCP RTO.

The Multimodal RTO algorithm is applicable for current and future retransmission timer-based protocols and applications to be used in LPWAN.

IX. Appendix

This appendix provides the details of the Peak-Hopper RTO and the CoCoA RTO algorithms [8, 9], and the settings assumed for these algorithms in this paper.

A. Peak-Hopper RTO

The Peak-Hopper RTO comprises a set of 5 steps. The first step determines the value of δ, a variable defined as follows:

$$\delta = (R - R_{previous}) / R_{previous} \quad \text{(Step } i\text{)}$$

where $R_{previous}$ denotes the previous RTT sample. In this paper, we assume an initial value for $R_{previous}$ of 1 s.

The second step computes a decay factor, D:

$$D = 1 - 1/(F \cdot S) \quad \text{(Step } ii\text{)}$$

where S is set to 1, since we assume sporadic message transmission, and F is set to 16 as proposed by the Peak-Hopper RTO authors [8]. Thus, in this paper D is 15/16.

The third step computes a booster factor, B:

$$B \leftarrow \min(\max(2 \cdot \delta, D \cdot B), B_{max}) \quad \text{(Step } iii\text{)}$$

where B_{max} is an upper limit to B, which is set to 1 in this paper since we assume that timestamps are not used [9]. We set the initial value of B to 1.

The fourth step determines R_{max}, defined as follows:

$$R_{max} = \max(R, R_{previous}) \quad \text{(Step } iv\text{)}$$

Finally, the RTO is computed as:

$$RTO \leftarrow \max(D \cdot RTO, (1+B) \cdot R_{max}, RTO_{min}) \quad \text{(Step } v\text{)}$$

where the RTO is chosen as the maximum among the three following terms: the first term decays the previous RTO, the second one corresponds to short RTT history, and RTO_{min} is equal to $R_{max}+2 \cdot G$. Considering the large RTT values in LPWAN, we assume $R_{max} \gg 2 \cdot G$, thus $RTO_{min} \approx R_{max}$.

B. CoCoA RTO

The CoCoA RTO defines two internal RTOs: the strong RTO and the weak RTO, each one based on the TCP RTO. When a strong or a weak RTT is measured, the corresponding RTO evolves accordingly. The most recently updated internal RTO, $RTO_{internal}$, contributes to the overall estimator, RTO_{output}, as shown next:

$$RTO_{output} \leftarrow a \cdot RTO_{internal} + (1-a) \cdot RTO_{output}$$

where $RTO_{internal}$ corresponds to the strong or the weak RTO. Default weights for a are 0.5 and 0.25 for the strong and for the weak RTO estimators, respectively. In this paper, we set a=0 for the weak estimator, to reduce CoCoA RTO underperformance in multimodal RTT scenarios.

References

[1] U. Raza et al., "Low Power Wide Area Networks: An Overview", IEEE Communications Surveys & Tutorials, vol. 19, no. 2, pp. 855-873, Second Quarter 2017.

[2] C. Gomez et al., "From 6LoWPAN to 6Lo: Expanding the Universe of IPv6-Supported Technologies for the Internet of Things", IEEE Communications Magazine, vol. 55, no. 12, pp. 148-155, Dec. 2017.

[3] S. Farrel, "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, May 2018.

[4] A. Minaburo et al., "LPWAN Static Context Header Compression (SCHC) and Fragmentation for IPv6 and UDP", draft-ietf-lpwan-ipv6-static-context-hc-24, Dec. 2019 (work in progress). Available at https://tools.ietf.org/html/draft-ietf-lpwan-ipv6-static-context-hc-24 (Accessed on 2020-01-04).

[5] C. Gomez et al., "IPv6 over LPWANs: Connecting Low Power Wide Area Networks to the Internet (of Things)", IEEE Wireless Communications (to be published).

[6] C. Gomez and J. Crowcroft, “RTO Considerations in LPWAN”, draft-gomez-lpwan-rto-considerations-01, Jul. 2019 (work in progress). Available at https://tools.ietf.org/html/draft-gomez-lpwan-rto-considerations-01 (Accessed on 2020-01-04).
[7] V. Paxson et al., “Computing TCP's Retransmission Timer”, RFC 6298, Jun. 2011.
[8] H. Ekstrom and R. Ludwig, “The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport”, Proc. IEEE INFOCOM 2004, vol. 4, Mar. 2004, pp. 2502–2513.
[9] A. Betzler et al., "CoAP Congestion Control for the Internet of Things", IEEE Communications Magazine, vol. 54, no. 7, pp. 154-160, July 2016.
[10] L. Casals et al., "Modeling the Energy Performance of LoRaWAN", Sensors, vol. 17, no. 10, Oct. 2017, Art. no. 2364.
[11] C. Gomez et al., "A Sigfox Energy Consumption Model", Sensors, vol. 19, no. 3, Feb. 2019, Art. no. 681.
[12] A. Minaburo and L. Toutain, “CoAP Time Scale Option”, draft-toutain-core-time-scale-00, Oct. 2017 (work in progress). Available at https://tools.ietf.org/html/draft-toutain-core-time-scale-00 (Accessed on 2020-01-04).
[13] J. Haxhibeqiri et al., “A Survey of LoRaWAN for IoT: From Technology to Application”, Sensors, vol. 18, no. 11, Nov. 2018, Art. no. 3995.
[14] A. Pop et al., “Does Bidirectional Traffic Do More Harm Than Good in LoRaWAN Based LPWA Networks?”, IEEE Global Communications Conference (GLOBECOM), Singapore, 2017, pp.1-7.
[15] J. Postel, “Transmission Control Protocol", RFC 793, Sep. 1981.
[16] V. Jacobson, “Congestion Avoidance and Control”, ACM SIGCOMM Computer Communication Review, vol. 18, no. 4, pp. 157-173, 1995.
[17] R. Braden, “Requirements for Internet Hosts–Communication Layers”, RFC 1122, Oct. 1989.
[18] V. Paxson, M. Allman, J. Chu, and M. Sargent, “Computing TCP’s Retransmission Timer", RFC 2988, Nov. 2000.
[19] A. Acharya and J. Saltz, “A Study of Internet Round-Trip Delay”, University of Maryland, Tech. Rep. CS-TR-3736, 1996.
[20] M. Allman, “Retransmission Timeout Requirements”, draft-ietf-tcpm-rto-consider-09, Dec. 2019 (work in progress). Available at: https://tools.ietf.org/html/draft-ietf-tcpm-rto-consider-09 (Accessed on 2020-01-04).
[21] M. Rajiullah et al., “An Evaluation of Tail Loss Recovery Mechanisms for TCP”, ACM SIGCOMM Computer Communication Review, vol. 45, no. 1, pp. 5-11, Jan. 2015.
[22] M. Allman and V. Paxson, “On Estimating End-to-end Network Path Properties”, ACM SIGCOMM Computer Communication Review, vol. 29, no. 4, pp. 263-274, Aug. 1999.
[23] I. Psaras and V. Tsaoussidis, “On the Properties of an Adaptive TCP Minimum RTO”, Computer Communications, vol. 32, no. 5, pp. 888-895, Mar. 2009.
[24] N. Seddigh and M. Devetsikiotis, “Studies of TCP's retransmission Timeout Mechanism”, IEEE ICC, vol. 6, Helsinki, Finland, 2001, pp. 1834-1840.
[25] P. Sarolahti and A. Kuznetsov, “Congestion Control in Linux TCP,” Proc. FREENIX Track: 2002 USENIX Annual Tech. Conf., Berkeley, CA, 2002, pp. 49–62.
[26] S. Bolettieri et al., “pCoCoA: A Precise Congestion Control Algorithm for CoAP”, Ad Hoc Networks, vol. 80, pp. 116-129, Nov. 2018.
[27] I. Jarvinen et al., “FASOR Retransmission Timeout and Congestion Control Mechanism for CoAP”, IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, UAE, 2018, pp. 1-7.
[28] Z. Shelby et al., “The Constrained Application Protocol (CoAP)”, RFC 7252, Jun. 2014.
[29] L. Kleinrock, “Queueing Systems. Volume I: Theory”, New York, NY, USA, Wiley Interscience, 1975, pp. 174-191.
[30] M. Lauridsen et al., “Interference Measurements in the European 868 MHz ISM Band with Focus on LoRa and SigFox”, 2017 IEEE Wireless Communications and Networking Conference (WCNC), San Francisco, CA, USA, 2017, pp. 1-6.