Benchmarking Buffer Size in IoT Devices Deploying REST HTTP

Cao Vien Phung, Mounir Bensalem and Admela Jukan
Technische Universität Braunschweig, Germany
{c.phung, mounir.bensalem, a.jukan}@tu-bs.de

***Abstract*—A few potential IoT communication protocols at the application layer have been proposed, including MQTT, CoAP and REST HTTP, with the latter being the protocol of choice for software developers due to its compatibility with the existing systems. We present a theoretical model of the expected buffer size on the REST HTTP client buffer in IoT devices under lossy wireless conditions, and validate the study experimentally. The results show that increasing the buffer size in IoT devices does not always improve performance in lossy environments, hence demonstrating the importance of benchmarking the buffer size in IoT systems deploying REST HTTP.**

***Index Terms*—REST HTTP; IoT; client; server; buffer size; Markov chain; node churn; benchmarking; wireless networks.**

I. Introduction

IoT (Internet of Things) systems use a few communication protocols to connect sensors and actuators with the IoT data and routing hubs, including MQTT (Message Queue Telemetry Transport) [1], CoAP (Constrained Application Protocol) [2] and REST HTTP (Representational State Transfer and Hypertext Transfer Protocol) [3]. The latter includes two layers, the upper layer being REST and the lower layer being HTTP. HTTP is the fundamental client-server model protocol, which is widely used by developers today due to its compatibility with existing network infrastructure [4]. The client-server model of HTTP protocol is performed by sending a request message to the server by the client, and then returning a corresponding acknowledgement back to the client by the server if that request message was accepted. REST is based on a specific architectural style with the guideline for web services developments to define the interaction among different components, and has been recently combined with HTTP protocol and deployed in IoT-based systems [4].

When using REST HTTP in IoT systems, the analysis of buffer size is critical because it directly impacts the communication performance, whereby larger size buffers are expected to reduce data packet blocking and thus increase transmission reliability. Also, larger buffers can constrain the hardware architecture and physical size of the IoT devices. Selecting an inappropriate buffer capacity is hence a tradeoff between communication performance and hardware design, requiring proper benchmarking of the buffer size. Previous works, such as [5] focused on some related important aspects such as interaction with underlying transport protocols, HTTP performance over satellite channels [6], and presentation of an approximate analytical model related to underlying transport layer [7]. Other works, such as [8] analyzed the impact of pipelining on the HTTP latency, while [9] and [10] analyzed the amount of redundant REST HTTP data in unstable environments, and yet [11] focused on the impact of HTTP pipelining. Of special interest is related work [12] which analyzed the receiving buffer size of IoT edge router (server), based on the analysis of the average TCP (Transmission Control Protocol) congestion window size of all IoT devices. This paper focused on scenarios where data streams arrive from a large number of IoT devices and experience packet losses due to congestion.

This paper benchmarks the buffer size in IoT client device deploying REST HTTP. We develop a novel analysis, under the assumption of IoT device availability churns, resulting in intermittent and unreliable communication. These events can cause network volatility with unusually high latency and fully closed connections, therefore retransmissions for timeout events set for RESTful applications are necessary with the aim of fault tolerance [13]. Our Markov chain analyzes this and derives the impact of message losses, caused by unusually high delay from node churns on expected buffer size on the client buffer of IoT devices. The analytical results are validated by the experiments in a simple testbed with IoT devices experiencing node churn. The results show increasing the buffer size in IoT devices does not always improve performance in lossy environments, in other words, that does not always decrease the amount of arrival data blocked at the client side, hence demonstrating the importance of proper buffer size benchmarking in IoT systems.

II. Analytical Model

We assume that IoT client devices experience the so-called node churn. The node churn means a device can appear and disappear from the system due to changed location, depletion of the battery or loss of wireless connectivity. Node churn can cause variable latency. Hence, it leads to generating automatically retry messages due to timeout events set for RESTful applications. In the example shown in Fig. 1, one constrained REST HTTP client device, e.g., sensor node, tries to add resources with POST methods on one REST HTTP server, e.g., edge node. The data transmission is considered under stop-and-wait mechanism. We assume that the arrival time interval t between two POST requests is constant, whereby t can be arbitrarily chosen. For example, each i^{th} arrival message q_i updated at a time at the client buffer with a constant arrival time interval is $t = 1$s. This example can be referred to real time monitoring applications, which need to periodically update the temperature and humidity data at a specific time interval [14]. Request message q_i is generated by the client REST layer using JSON file format, which is often used in IoT standards over HTTP. Before transmitting the JSON file of q_i, HTTP layer adds a header to POST method, which contains the HTTP version, the Content-type, and the root of the resource, etc. Similarly as [14], we assume that the content of messages has the same length. For RESTful applications with fault tolerance, timeout interval T_o is necessary [13], and it is arbitrarily chosen. In this example, we set $T_o = 2$s. Arrival messages are stored at the client REST buffer and only deleted when they are successfully sent and their corresponding responses are successfully returned. The authors [13] gave an example of five-time retransmission for each message failed, however, we assume unlimited retransmissions in this paper.

We define the *OBSERVATIONS* as request message read processes right after either timeout event terminated or arrival of the first request message event when the client buffer is empty. Hence, in our assumption there always exists at least one message in the buffer. The client buffer capacity M is defined to be the maximum number of messages that the client can buffer. We assume $M = km + 1$, whereby k is a constant arbitrarily chosen, we only consider $k \in \{1, 2\}$, and $m = \frac{T_o}{t} \in \mathbb{N}^*$ is the number of arrival request messages updated at the client side in a timeout event. For instance, in Fig. 1 we choose $k = 2$ and $m = \frac{T_o}{t} = 2$, i.e., the client has the buffer capacity $M = 5$ messages. Assume that new request messages arrived at the client are ignored, when previous request messages stored in the client buffer are successfully sent in case of smooth connections. However, the amount of ignored arrival data in this assumption is extremely small. This can be explained by the fact that the time interval t between two request messages arrived at the client is much higher than the transmission time of stored request messages with smooth connections, whereby theoretical and experimental results presented in the next section confirm our explanation.

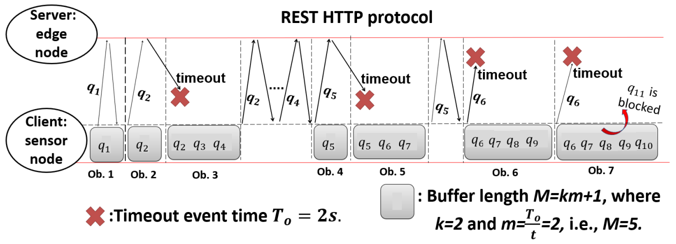


Figure 1: REST HTTP scenario in lossy environment, where Ob. j is observation j, q_i is message i and $t = 1$s is arrival time interval of each message updated at a time at the client.

In Fig. 1, message q_1 arriving at Ob. 1 (Ob. j denotes Observation j for short) is removed from the client buffer, when its response is successfully returned on the first try. For q_2 arriving at Ob. 2, assuming its response is lost on the way back to the client, the client keeps buffering this message along with $m = 2$ arrival messages of q_3 and q_4, after timeout event terminated at Ob. 3. Assume that any timeout event occurs due to network volatility causing fully closed connections. When q_2, q_3 and q_4 are removed, q_5 stored in the buffer arrives at its arrival time of Ob. 4. Similarly, assuming the response of q_5 is lost on the way back to client, the client keeps buffering it along with $m = 2$ arrival messages of q_6 and q_7, after timeout event terminated at Ob. 5. q_5 is only removed from the buffer when its response is successfully returned in the second attempt. When q_6 is lost, 4 messages (q_6-q_9) are buffered after its first timeout event terminated at Ob. 6. As $M = 5$, q_{10} is added into the buffer and q_{11} is blocked/lost, i.e., q_6 to q_{10} are buffered, after the second timeout of q_6 terminated at Ob. 7.

Table I: List of notations.

Notation	Meaning
p	Failed message probability.
m	Arrival messages updated at the client in a timeout event.
$\pi(e)$	Steady-state probability of e requests.
$\delta_{u\to v}$	Transition probability from state u to state v.
M	Client buffer capacity.
S	Expected client buffer size.
T_o	Time interval for a timeout event.
t	Arrival time of each message updated at a time at the client.

A. State transition probability of Markov chain model

In this subsection, we analyze the state transition probability of Markov chain model, presented in Fig. 2.

1) $k = 2$: Fig. 2 is the Markov chain with $k = 2$ reflecting the above defined scenario, whereby p is the failed message probability. The state space with $k = 2$ is $E = \{1, m+1, m+2, ..., 2m+1\}$, including $m+2$ states, where any state denotes the number of requests stored in the buffer. For example, the state $m + 1$ means the client buffers $m + 1$ messages. The notations are summarized in Table I. The transition probability from state $u = 1$ to state $v = 1; m + 1$ is:

$$\delta_{1\to v} = \begin{cases} 1-p & , v = 1 \qquad (1a) \\ p & , v = m+1 \qquad (1b) \end{cases}$$

Eq. (1a) shows state 1 keeps staying itself, when only one message is stored in the buffer and it is successfully sent with the transition probability $\delta_{1\to 1} = 1 - p$. Since we observe arrival of the first message event when the client buffer is empty, there is no state of 0. For example, in Fig. 1 after q_1 arriving at Ob. 1 is successfully sent, only q_2 stored in the buffer arrives at its arrival time of Ob. 2, i.e., state 1 remains after Ob. 1 and Ob. 2 with transition probability $\delta_{1\to 1} = 1-p$.

Eq. (1b) shows state $u = 1$ directly transits to state $v = m+1$, when only one message is buffered and its timeout event occurs with $\delta_{1\to m+1} = p$. In Fig. 2, based on the observation defined above, there are no direct state transitions from state 1 to state $v \in [m+2, 2m+1]$ and state $v \in (1, m]$ does not exist. The reason is that the buffer increases the maximum number of m arrival messages at the client after each timeout event terminated, while as assumed at least one message is always buffered. For instance, in Fig. 1 as observed for timeout event terminated at Ob. 3, there are 3 messages in the buffer, i.e., state $u = 1$ at Ob. 2 directly transits to state $v = m + 1 = 3$ at Ob. 3, where $m = 2 < v = m + 1 = 3 < m + 2 = 4$. In other words, there is no state $m = 2$ and there are no direct state transitions from state $u = 1$ to state $v \in \{4; 5\}$ in Fig. 1.

The transition probability from state $u \in [m+1, 2m+1]$ to state $v = 1 \bigcup [m+1, 2m+1]$ is given by:

$$\delta_{u \to v} = \begin{cases} (1-p)^u & , v = 1 \quad \text{(2a)} \\ (1-p)^{u-v+m} p & , v \in [m+1, 2m] \quad \text{(2b)} \\ \sum_{i=0}^{u+m-v} (1-p)^i p & , v = 2m+1 \quad \text{(2c)} \end{cases}$$

Eq. (2a) denotes state $u \in [m+1, 2m+1]$ directly transits to state 1, when all messages in the buffer are successfully sent with $\delta_{u \to 1} = (1-p)^u$. For example in Fig. 1, $q_2 - q_4$ buffering at Ob. 3 are removed and q_5 stored in the buffer arrives at its arrival time of Ob. 4, i.e., state $u = m+1 = 3$ at Ob. 3 transits to state $v = 1$ at Ob. 4 with $\delta_{3 \to 1} = (1-p)^3$.

Eq. (2b) indicates that state $u \in [m+1, 2m+1]$ can directly transit to state $v \in [m+1, 2m]$. The oldest $u - v + m$ consecutive messages are successfully sent with probability $(1-p)^{u-v+m}$, but $(u-v+m+1)^{th}$ message is failed with loss probability p. Therefore, the transition probability from state u to state v in this case is $\delta_{u \to v} = (1-p)^{u-v+m}p$, e.g., in Fig. 2, $\delta_{2m \to m+2} = (1-p)^{2m-2}p$. As previously discussed, there is no state $v \in (1, m]$ and there are also no state transitions from $u \in [m+1, 2m+1]$ to state $v \in (1, m]$. For example in Fig. 1, one message q_5 belonging to Ob. 5 is successfully removed at the client, and the client buffers 4 messages after the timeout event of q_6 terminated at Ob. 6, i.e., state $u = m+1 = 3$ at Ob. 5 directly transits to state $v = m+2 = 2m = 4$ $(4 > m = 2)$ at Ob. 6 with $\delta_{3 \to 4} = (1-p)p$.

Eq. (2c) shows that state $u \in [m+1, 2m+1]$ directly transits to state $v = 2m+1$ with transition probability $\delta_{u \to 2m+1} = \sum_{i=0}^{u-m-1} (1-p)^i p$. Since the client buffer capacity is limited to $M = 2m+1$, any state u under this consideration cannot transit to states larger than $2m+1$. Hence, a few messages are blocked at the client, if the total number of request messages need to be stored more than $2m+1$ of buffer capacity M. For instance, in Fig. 2, the transition probability from state $u = m+2$ to state $v = 2m+1$ is $\delta_{m+2 \to 2m+1} = p + (1-p)p$. In this example, if the oldest message fails, or it succeeds but the second oldest one fails, then the state of client buffer $u = m+2$ always transits to state $v = 2m+1$. The reason is that the client buffer always increases the maximum number of m arrival request messages after a timeout event terminated, but the client buffer cacpacity is limited to $M = 2m+1$. For this example, one arrival request message is blocked at the client after terminated timeout event in case of the oldest request message unsuccessfully sent. This blocking event can be explained that after the terminated timeout event the client buffer needs to store $2m+2$ request messages, while the client buffer capacity is limited to $M = 2m+1$. Let's take the example in Fig. 1, after the terminated timeout event at Ob. 7, as $M = 2m+1 = 5 < 6$, message q_{11} is blocked, i.e., state $m+2 = 4$ at Ob. 6 directly transits to state $2m+1 = 5$ at Ob. 7. On the other hand, no message is blocked in case of the oldest request message successfully sent but the second oldest one failed because the client buffer removes the oldest one after its success. For the latter case, the buffer capacity M is sufficient to store $2m+1$ messages. We note that the transition probability from state $u = m+1$ to state $v = 2m+1$ can be also applied by Eq. (2b), i.e., $\delta_{m+1 \to 2m+1} = p$.

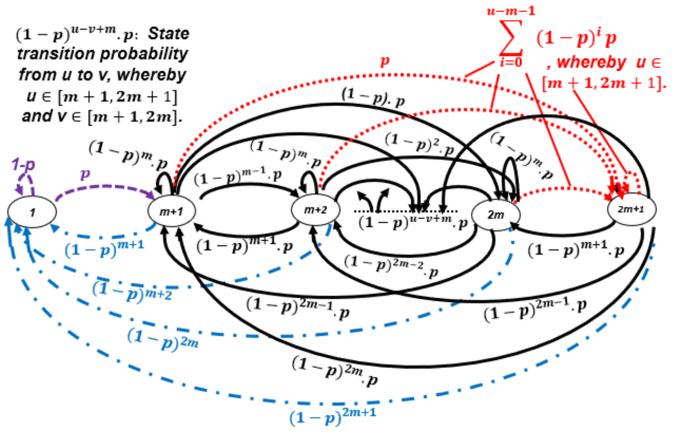


Figure 2: Markov chain model with $k = 2$ for the client buffer with REST HTTP in lossy IoT system.

2) $k = 1$: With $k = 1$, the state space only includes two states to be $E = \{1, m+1\}$. The transition probability from 1 to itself and $m+1$ are Eq. (1a) and Eq. (1b), respectively. The transition probability from the state $m+1$ to 1 and itself are Eq. (2a) and Eq. (2c), respectively.

B. Lemmas and theorems based on the Markov chain model

Lemma 1. *The Markov chain model with $k = 1$ and $k = 2$ is finite, irreducible and aperiodic.*

Proof. The state space of these Markov chains is finite as it has 2 states for $k = 1$ and $m+2$ states for $k = 2$. Additionally, all states can communicate together. More specially, for $k = 1$, the two states of 1 and $m+1$ can be reached together with a non-zero probability after 1 transition step. Also, for $k = 2$ in Fig. 2, a state $u \in [m+1, 2m+1]$ and $u = 1$ are reachable from any other state $v \in \{1, m+1, m+2, ..., 2m+1\} \setminus \{u\}$ and $v = m+1$, respectively, with a non-zero probability after 1 transition step. In addition, $u = 1$ is reachable from any other state $v \in [m+2, 2m+1]$ with a non-zero probability after 2 transition steps in Fig. 2. Hence, these Markov chains are irreducible. Finally, we prove that these Markov chains are aperiodic by proving that any state u is aperiodic. The period $d(u)$ of a state u is the greatest common denominator of all integers $n > 0$, for which the $n-$step transition probability from state u to itself $P_{uu}^{(n)} > 0$. As each state of these Markov chain models has a self-transition with a non-zero probability, $d(u) = 1$ and consequently state u is aperiodic. □

As the Markov chains of REST HTTP in lossy environments with $k \in \{1, 2\}$ satisfy Lemma 1, they have a steady-state distribution to which the distribution converges, starting from any initial state. The total probability of all steady-states is $\sum_{e \in E} \pi(e) = 1$, whereby $\pi(e)$ is the steady-state probability of e messages in the buffer. With section II-A, we build the balance equations of any state defined that the total probability of entering it is equal to the total probability of leaving it.

Case of $k \geq 1$ for the state 1, based on the analysis in section II-A and using Eq. (1b) and Eq. (2a), we obtain the balance equation Eq. (3) of state 1 for use case $k \geq 1$:

$$\pi(1) \cdot p = \sum_{i=m+1}^{km+1} \pi(i) \cdot (1-p)^i \tag{3}$$

1) For $k = 1$: Eq. (3) is the balance equation of state 1. Based on the analysis in section II-A and using Eq. (1b) and Eq. (2a), we obtain the balance equation Eq. (4) of state $m+1$:

$$\pi(m+1) \cdot (1-p)^{m+1} = \pi(1) \cdot p \tag{4}$$

Lemma 2. *For $k = 1$, the probability of steady-states is:*

$$\pi(1) = \frac{(1-p)^{m+1}}{p + (1-p)^{m+1}} \ ; \ \pi(m+1) = \frac{p}{p + (1-p)^{m+1}} \tag{5}$$

Proof. We use Eq. (4) and $\sum_{e \in E} \pi(e) = 1$ to solve. □

Theorem 1. *The expected buffer size S is the average of the number of messages stored in the buffer. For $k = 1$, S is given:*

$$S = \frac{(1-p)^{m+1} + (m+1)p}{p + (1-p)^{m+1}} \tag{6}$$

Proof. We use lemma 2 and $S = \sum_{e \in E} e\pi(e)$ to solve. □

2) For $k = 2$: Eq. (3) with $k = 2$ is the balance equation of state 1. The balance equation of state $m+1$ for $k = 2$ is referred to Appendix A, and it is simplified by using Eq. (3):

$$\pi(m+1) = \pi(1) \cdot p(1-p)^{-1} \tag{7}$$

For all $e \in [m+2, 2m]$, the balance equations of state e for the case of $k = 2$ can be referred to Appendix B and Appendix C, and then they are simplified by using Eq. (3) with $k = 2$:

$$\pi(e) = \pi(1) \cdot p^2 (1-p)^{m-e} \tag{8}$$

The balance equation of state $2m+1$ for the case of $k = 2$ can be referred to Appendix D, and then it is simplified by applying Eq. (3) with $k = 2$ and $\sum_{e \in E} \pi(e) = 1$:

$$\pi(2m+1) = 1 - [1 + p(1-p)^{-m}]\pi(1) \tag{9}$$

Lemma 3. *Let $r = 1 - p$ and $\breve{D} = p - r^m[p + (m-1)p^2 - r^{m+1} - pr]$, for $k = 2$, the probability of steady-states is:*

$$\pi(1) = \frac{r^{2m+1}}{\breve{D}} \ ; \ \pi(m+1) = \frac{pr^{2m}}{\breve{D}} \ ; \ \pi(e) = \frac{p^2 r^{3m-e+1}}{\breve{D}} \ ;$$

$$\pi(2m+1) = \frac{p[1 - r^m(1 + (m-1)p)]}{\breve{D}} \ ; \ \forall m+2 \leq e \leq 2m$$

Proof. We use Eqs. (3), (7), (8) and (9) to solve. □

Theorem 2. *$r = 1 - p$, the expected buffer size S for k=2 is:*

$$S = \frac{r^m[r(2r^m - 1) - mp^2(2m+1)] + (2m+1)p}{p - r^m[p + (m-1)p^2 - r^{m+1} - pr]}, \tag{10}$$

Proof. We use $S = \sum_{e \in E} e\pi(e)$ and Lemma 3 to solve. □

Corollary 1. *For all k, the limit of expected buffer size S is:*

$$\lim_{p \to 0} S = 1 \ ; \ \lim_{p \to 1} S = km + 1 \tag{11}$$

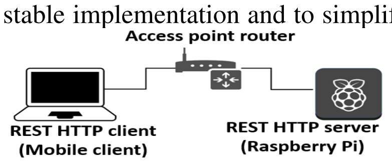
Proof. We take Theorem 1 and 2 to solve, where $p \to 0; 1$. □

3) For $k \geq 3$: With different values of k, we always have different functions of expected client buffer size S and probability of steady-states $\pi(e)$. This requires complex and major repeating calculations for each different value of k.

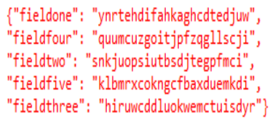
III. Validation and Benchmarking

A. Experimental setup

The experimental validation setup consists of a simple wired network for communication between a client and a server with REST HTTP (Fig. 3a). It includes a laptop (Intel(R) Core(TM) i7-3667U CPU @ 2.00GHz) representing our sensor node, a Raspberry Pi 3 Model B+ (64-bit quad-core ARM Cortex-A53 processor @ 1.4GHz and 1 GB of RAM) representing the static server running python flask version 1.0.3, and an access point router used to connect the laptop with the Raspberry Pi. The sensor node can be replaced by the laptop because the processing time of messages on devices is extremely small, compared with the arrival time interval t between two messages and with timeout time T_o, when our analytical model mainly focuses on these two kinds of time. Therefore, the analytical results of client buffer are not affected with using the laptop or sensor node. We consider a wired network to obtain a stable implementation and to simplify the setup. In unreliable



(a) An experimental wired network setup.



```
{"fieldone": "ynrtehdifahkaghcdtedjuw",
"fieldfour": "quumcuzgoitjpfzqgllscji",
"fieldtwo": "snkjuopsiutbsdjtegpfmci",
"fieldfive": "klbmrxcokngcfbaxduemkdi",
"fieldthree": "hiruwcddluokwemctuisdyr"}
```

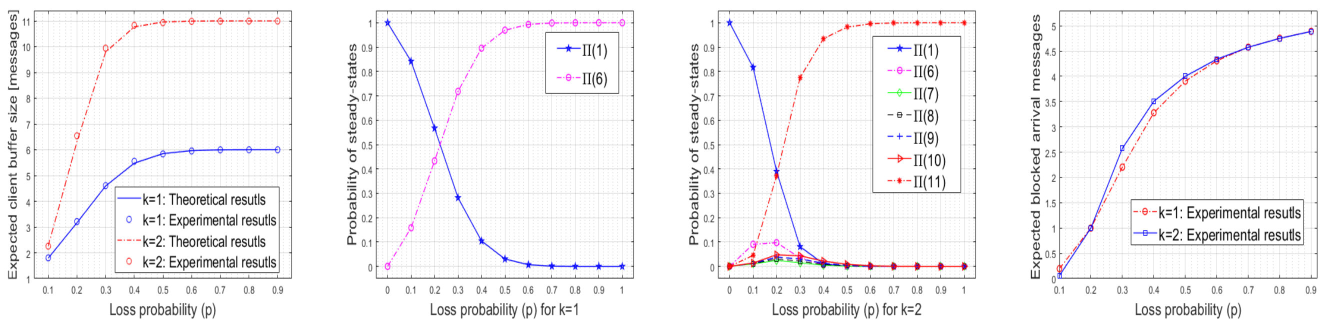
(b) JSON file [10]

Figure 3: Experimental setup.

Algorithm 1 Emulation of link loss until timeout expires [10]

```
1: procedure SOLVE($Link\_loss\_vector$)
2:     if we meet 1 from $Link\_loss\_vector$ then
3:         Pause ($t^{,}$), e.g., $t^{,} = timeout$
4:     end if
5: end procedure
```

environments, we consider stop-and-wait mechanism and the number of retransmissions unlimited. In order to emulate the connection losses of the sensor node in our wireless scenario, we incorporate them into the experimental setup using Algorithm 1 [10], where p denotes the message loss probability. An example for this emulation is as follows: We randomly generate a vector of binary numbers, where 20% of the values are ones, e.g., $vector = (0, 0, 0, 1, 0)$: 1 represents the connection loss until expired timeout and 0 represents no connection losses. The proposed example means that 20% of links are lost. Using the link loss vector for Algorithm 1, if the client or server meets the value of 1, then the transmission is paused for a period of time $t^{,}$ to represent a connection loss. $t^{,}$ should be set so that it is equal to the value of $timeout$ T_o set for the client or $t^{,}$ should be long enough so that the response message cannot be returned to the client side. Algorithm 1 is required to be executed before dispatching the request and response message. When timeout expires, the client forgets the previous connection and opens a new one for retransmission.



(a) Theory and experiment of expected buffer size S for $k = 1; 2$.

(b) Theory of probability of steady-states for $k = 1$.

(c) Theory of probability of steady-states for $k = 2$.

(d) Experiment of expected blocked arrival messages for $k = 1; 2$.

Figure 4: Theoretical and experimental results for $k = 1$ and $k = 2$.

The client uses python request library and protocol HTTP/1.1 to send POST messages to the server. A POST $message = requests.post('http://192.168.1.2:5000/api/sensors', timeout = T_o, data = POST\_Data)$ is sent to the Raspberry Pi, where 192.168.1.2 is Raspberry Pi's IP address, timeout event is $T_o = 15s$ and $POST\_Data$ is the JSON file randomly created, as shown in Fig. 3b. The arrival time interval of each message updated at a time at the client is set $t = 3s$. Hence, the number of arrival messages at the client during timeout is $m = \frac{T_o}{t} = 5$ messages. Each message has the header $\{Header : ID\}$, e.g., message q_i has $ID = i$. The message length is 199 bytes. The client buffer capacity is $M = km + 1$ messages, where $M = 6$ messages and $M = 11$ messages are for k=1 and for k=2, respectively. Each time period of 8.3 hours was observed to experimentally measure for each result point in Fig. 4a and Fig. 4d.

B. Evaluation

Fig. 4a presents the theoretical and experimental results of expected buffer size S for $k = 1$ ($M = 6$) and $k = 2$ ($M = 11$), vs. message loss probability p. Theorem 1 and Theorem 2 are used to draw the theoretical results for $k = 1$ and $k = 2$. For $0 \leq p \leq 0.5$, the expected buffer size S obtained by the theory and experiment increases with increasing p because the buffer stores more messages, when p increases. Nevertheless, when $p \geq 0.5$, S approximately achieves the constant value of M because the buffer is often full in this loss probability interval. The larger the buffer length M is, the higher the expected buffer size S is because a higher value M has a greater storage capacity. Based on corollary 1, we have $\lim_{p\to 0} S = 1$ for all k, $\lim_{p\to 1} S = 6$ for $k = 1$ and $\lim_{p\to 1} S = 11$ for $k = 2$. For $p < 0.5$, there is a small difference between the theoretical and experimental results. This difference increases in terms of the value k. For example of $k = 1$ at $p = 0.2$, $S = 3.164$ messages are for theory and $S = 3.211$ messages are for experiment (experiment increases by 1.49%). Also, for $k = 2$ at $p = 0.2$, $S = 6.302$ messages are for theory and $S = 6.540$ messages are for experiment (experiment increases by 3.78%). The reason of the insignificant difference is that: For theory, we assume that new request messages arrived at the client are ignored, when previous request messages stored in the client buffer are successfully sent in case of smooth connections. This can be explained by the fact that the mean arrival time $t = 3$s is much higher than the transmission time of stored messages. Let's take an example in table II for 10 messages stored in the buffer, we only consume 0.161s to successfully send them with smooth connections. Therefore, the ignored data in our assumption is insignificant, even approximately equal to 0. For $p \geq 0.5$, the theoretical and experimental results are mostly overlapped because almost new messages arrived at the client are from timeout events, which the experimental model fits with the theoretical one. Fig. 4b and Fig. 4c present theoretical results of steady-states probability with buffer capacity M of $k = 1$ and $k = 2$, respectively, vs. message loss probability p. Lemma 2 and Lemma 3 are used for Fig. (4b) and Fig. (4c), respectively. For the steady-state probability $\pi(1)$ in Fig. (4b) and Fig. (4c), the higher the loss probability p is, the lower it is. For the steady-state probability $\pi(km + 1)$, i.e., $\pi(6)$ and $\pi(11)$ in Fig. (4b) and Fig. (4c), respectively, the higher the loss probability p is, the higher it is. For $\pi(e)$ ($e \in [6, km]$) in Fig. 4c, $\pi(e)$ gradually increases and then tends to gradually decrease to 0. The reason for those behaviors is the number of arrival messages stored in the buffer increases if p increases.

Table II: Examples of experimental transmission time for successfully sending messages with smooth connections.

Number of messages stored in the buffer	Total time in seconds
1	0.034
3	0.048
5	0.089
8	0.122
10	0.161

Next, we analyse the steady-state probability $\pi(km + 1)$ in Fig. 4b and Fig. 4c to understand the impact of the value k on the expected blocked arrival messages B at the client, i.e., $\pi(6)$ in Fig. 4b and $\pi(11)$ in Fig. 4c. For $p \in [0.1; 0.2]$, we obtain $\pi(6) > \pi(11)$, e.g., when $p = 0.1$, we have $\pi(6) = 0.158$ and $\pi(11) = 0.045$. This can be explained by the fact that with a higher value k we have a higher number of states, therefore we obtain a lower value $\pi(km+1)$. For $p \in [0.6; 1]$ we obtain the same value $\pi(km + 1)$, e.g., $\pi(6) \approx \pi(11) \approx 1$, because with a very high loss probability p, the client buffer is always full in this probability interval. In the interval of comparatively high value $p \in [0.3; 0.5]$, we have $\pi(6) < \pi(11)$, e.g., $\pi(6) = 0.718$ and $\pi(11) = 0.774$ at $p = 0.3$. In this case, the storage capacity of new arrival messages increases with the increase of the value k, i.e., increasing transmission chances of messages leads to increasing timeout events in this probability interval, which can increase the number of blocked arrival messages if increasing k. In Fig. 4d, we show experimentally the expected blocked arrival messages B at an observation at the client. The observation is defined in section II, $\lim_{p\to 0} B = 0$, and $\lim_{p\to 1} B = m = 5$ messages, i.e., 5 messages arrived at the client are blocked in the timeout observation event when p achieves 1. With $p \in [0.3; 0.5]$ in Fig. 4d, B of $k = 2$ is higher than B of $k = 1$, which confirms the theoretical analysis above (impact of the value k on B, based on the analysis of steady-state probability $\pi(6)$ in Fig. 4b and $\pi(11)$ in Fig. 4c). B of $k = 1$ is approximately equal to B of $k = 2$ for $p > 0.5$, while B of $k = 2$ is less than B of $k = 1$ for $p = 0.1$. With those analyses, we recommend setting $k = 2$ if $p < 0.2$ and $k = 1$ if $p \geq 0.2$ for reducing the blocked data.

Through the analysis above, we can derive an interesting conclusion that a large buffer capacity M for reliable IoT systems in lossy environments does not always yield the best performance. Here, the amount of blocked data can increase, since a large M value can cause more timeout events leading to increasing the number of arrival messages at the buffer, while the buffer is already filled with a large number of messages. This is consistent with discussions showed in [12] for TCP.

IV. Conclusion

We benchmarked the buffer size in IoT devices deploying REST HTTP, in theory and experiment. The results showed that a large buffer in IoT devices does not always lead to an improved performance in lossy environments, and in fact could even degrade the performance. The proper benchmaking of the buffer size is hence rather important. The experimental analysis indicated that in order to reduce the blocked data, we should set $k = 2$ if $p < 0.2$ and $k = 1$ if $p \geq 0.2$. In future work, we plan to analyze more complex cases for $k \geq 3$.

References

[1] OASIS, "MQTT Version 3.1.1," *OASIS Standard*, p. 81, 2014.

[2] Z. Shelby and C. Hartke, K. Bormann, "rfc7252, The Constrained Application Protocol (CoAP)," pp. 1–112, 2014.

[3] C. Severance, "Roy T. Fielding: Understanding the REST Style." *Computer*, vol. 48, no. 6, pp. 7–9, 2015.

[4] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Comput. Surv.*, vol. 51, no. 6, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3292674

[5] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the performance of http over several transport protocols," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 616–630, Oct. 1997.

[6] H. Kruse, M. Allman, J. Griner, and D. Tran, "Experimentation and modelling of http over satellite channels," *Inter. Jour. of Satellite Communications*, vol. 16, no. 1, pp. 51–68, 2001.

[7] P. Vaderna, E. Stromberg, and T. Elteto, "Modelling performance of http/1.1," in *GLOBECOM '03*, vol. 7, Dec 2003, pp. 3969–3973 vol.7.

[8] W. Bziuk, C. V. Phung, J. Dizdarevic, and A. Jukan, "On http performance in iot applications: An analysis of latency and throughput," in *2018 MIPRO*, May 2018, pp. 0350–0355.

[9] C. V. Phung, J. Dizdarevic, F. Carpio, and A. Jukan, "Enhancing rest http with random linear network coding in dynamic edge computing environments," in *2019 MIPRO*, May 2019, pp. 435–440.

[10] C. V. Phung, J. Dizdarevic, and A. Jukan, "An experimental study of network coded rest http in dynamic iot systems," in *ICC 2020*, pp. 1–6.

[11] K. Shuang, T. Zhang, Z. Dong, and P. Xu, "Impact of http pipelining mechanism for web browsing optimization," in *2015 IEEE International Conference on Mobile Services*, June 2015, pp. 415–422.

[12] J. Khan, M. Shahzad, and A. Butt, "Sizing buffers of iot edge routers," 06 2018, pp. 55–60.

[13] J. Edstrom and E. Tilevich, "Reusable and extensible fault tolerance for restful applications," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012.

[14] N. A. M. Alduais, J. Abdullah, A. Jamil, and L. Audah, "An efficient data collection and dissemination for iot based wsn," in *IEMCON*, 2016.

Appendix

A. *Balance equation of state $m + 1$ for the case of $k = 2$*

Based on section II-A and using Eq. (1b), Eq. (2a) and Eq. (2b), the balance equation (12) of state $m + 1$ for $k = 2$ is:

$$\begin{aligned}\pi(m+1)\cdot\left[(1-p)^{m+1} + p\sum_{i=m+2}^{2m+1}(1-p)^{2m+1-i}\right]\\ = \pi(1)\cdot p + p\sum_{i=m+2}^{2m+1}\pi(i)\cdot(1-p)^{i-1}\end{aligned} \tag{12}$$

B. *Balance equation of state $e \in [m + 2, 2m - 1]$ for $k = 2$*

Based on section II-A and using Eq. (2a), Eq. (2b) and Eq. (2c), the balance equations (13) of state e for $k = 2$ is:

$$\begin{aligned}&\pi(e)\left[(1-p)^e + p\left(\sum_{i=m+1}^{e-1}(1-p)^{e-i+m}\right.\right.\\ &\left.\left.+\sum_{i=e+1}^{2m}(1-p)^{e-i+m} + \sum_{i=0}^{e-m-1}(1-p)^i\right)\right]\\ &= \sum_{i=m+1}^{e-1}\pi(i)\cdot(1-p)^{i-e+m}p + \sum_{i=e+1}^{2m+1}\pi(i)\cdot(1-p)^{i-e+m}p\end{aligned} \tag{13}$$

C. *Balance equation of state $2m$ for $k = 2$*

Based on section II-A and using Eq. (2a), Eq. (2b) and Eq. (2c), the balance equation (14) of state $2m$ for $k = 2$ is:

$$\begin{aligned}&\pi(2m)\left[(1-p)^{2m} + p\left(\sum_{i=m+1}^{2m-1}(1-p)^{3m-i} + \sum_{i=0}^{m-1}(1-p)^i\right)\right]\\ &= \left[\sum_{i=m+1}^{2m-1}\pi(i)\cdot(1-p)^{i-m}p\right] + \pi(2m+1)\cdot(1-p)^{m+1}p\end{aligned} \tag{14}$$

D. *Balance equation of state $2m + 1$ for $k = 2$*

Based on section II-A and using Eq. (2a), Eq. (2b) and Eq. (2c), the balance equation (15) of state $2m + 1$ for $k = 2$ is:

$$\begin{aligned}\pi(2m+1)\left[(1-p)^{2m+1} + p\sum_{i=m+1}^{2m}(1-p)^{3m-i+1}\right]\\ = \sum_{i=m+1}^{2m}\pi(i)\cdot\sum_{j=0}^{i-m-1}(1-p)^j p\end{aligned} \tag{15}$$