Open Access

Open Journal of Internet of Things (OJIOT)
Volume 5, Issue 1, 2019
http://www.ronpub.com/ojiot
ISSN 2364-7108

Data Lifetime Estimation in a Multicast-Based CoAP Proxy

Jelena Mišić [A], Vojislav B. Mišić [A], Xiaolin Chang [B]

[A] Ryerson University, 350 Victoria St., Toronto, ON, Canada, {jmisic, vmisic}@ryerson.ca
[B] Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University,
Shangyuancun no. 3, Haidian District, Beijing, China, xlchang@bjtu.edu.cn

Abstract

In this work we consider kernel-based record lifetime estimation in a proactive Internet of Things (IoT) proxy with multicast based cache management. Multicast refreshment requests were based on lifetime expiration for a predefined number of records. To reduce the traffic volume in the IoT domain, we assume that only nodes where the observed physical variable has changed its value will respond to the multicast request. For estimating the data lifetime at the proxy, we use Gaussian kernels, assuming that the intrinsic data lifetime probability distribution was taken from Erlang-k family of sub-exponential distributions. In this setup, we consider that the proxy connects to the IoT domain using an IEEE 802.15.4-compatible wireless network. Results indicate that narrow and symmetrical lifetime probability distributions require more frequent multicasting refreshments compared to wider and asymmetric ones. This increases traffic intensity and energy consumption in IoT domain. We quantify finding with numerical results.

Type of Paper and Keywords

Regular research paper: *CoAP, multicast proxy, cache maintenance, kernel-based estimation, data lifetime*

1 Introduction

The introduction of Constrained Application Protocol (CoAP) [15] has led to the proliferation of versatile applications over a large number of smart devices. CoAP acts as the application layer protocol that works over the UDP/IP protocol stack which facilitates evolution and integration of sensor and actuator networks to bring the Internet-of-Things (IoT) vision to life. CoAP supports this vision through asynchronous message exchange with optional reliability, publish-subscribe paradigm, multicasting, and resource discovery in both IPv4 and IPv6-based network environments. It thus allows the ubiquitous Web paradigm in which servers provide content to Internet clients to be replicated in an IoT network environment comprised of smart nodes that provide sensed data on demand to clients from the Internet [6].

However, IoT networks and applications face a set of challenges that differ from those encountered in traditional web scenarios. First, IoT server nodes have limited computational capabilities and they typically operate on battery power; this severely constrains the choices available to the designers of IoT networks. Second, most IoT applications require a certain level of freshness of sensed data in order to function correctly; this complicates the data collection protocols. Finally,

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2019)* in conjunction with the VLDB 2019 conference in Los Angeles, USA. The proceedings of VLIoT@VLDB 2019 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

allowing IoT server nodes to be contacted directly by clients opens security vulnerabilities. All of those challenges can be addressed through the use of proxies that interconnect low power, low data rate wireless networks of IoT devices to other, higher capacity wireless or even wired networks and provides the necessary protocol translation. The clients thus see the proxy as a traditional web server where the data is actually created by the IoT servers behind it.

The proxy typically maintains a cache to store data readings sent by the IoT server nodes. The use of cache reduces latency for the clients, but also saves communication bandwidth in the IoT domain and energy expenditure of IoT server nodes. In the simplest case, proxy will respond to client requests with cached data, if it is fresh, or request data from the appropriate IoT server node, if not. In the proactive mode, proxy will monitor the freshness of cached data and issue requests to refresh it when needed, without an explicit client request. The proxy can also operate in reverse mode in which IoT server nodes send data to the proxy when the data becomes stale (i.e., obsolete) or when the monitored physical variable changes. Oftentimes these operation modes are combined in the so-called hybrid proxies.

In all cases, the notion of data freshness is crucial for the timely operation of the proxy. To support monitoring of data freshness, CoAP allows IoT server nodes to annotate the data they send with a `Max_Age` value indicating the data lifetime. This allows the proxy to decide whether to service a client's request with cached data or to request the data from the server node. However, in many scenarios sensed data is parameterized: for example, instead of periodically reporting the exact data reading, the server node can report whether the value has crossed some threshold. This approach reduces the number of messages exchanged between the IoT server and the proxy, and further reduces communication bandwidth and energy.

In both periodic and parameterized reporting, the proxy can use the data lifetime value to refresh its cache. However, parameterization turns the data lifetime into a random variable and effectively precludes the server node from providing information about data lifetime. The latter case requires estimation of the probability distribution of the data lifetime. Estimation has to be done at the proxy since server nodes typically do not possess the required computational resources [19]. The use of multicasting [12] enables the proxy to wait until an entire group of data records become stale and refresh them using a single multicast GET request, which was found to be most efficient in terms of communications bandwidth and latency [7].

The focus of the present work is lightweight data collection and cache refreshment by a CoAP proxy operating in reverse mode, with the objective of reducing the volume of wireless traffic on the IoT side. It builds upon our previous work [8] but extends it significantly in several directions, as follows.

- We assume that data lifetime follows a subexponential probability distribution from Erlang-k family which is more realistic than the exponential distribution used in the earlier work.
- We deploy data lifetime estimation at the proxy and use it to estimate when data record might become stale i.e. when it exceeds threshold value.
- The proxy performs data lifetime estimation using kernel smoothing [17] which is a computationally efficient technique to find structure in data sets without a predefined parametric model.
- Data coming from IoT nodes present a group-based observation stream combining the concepts of multicasting [12] and observations [3]. Namely, multicast requests are sent when a number of cached data values exceed their estimated lifetime threshold, and IoT server nodes reply to multicast requests after a random leisure period. As the result, this approach provides adhoc and implicit multicast subgroups within the IoT domain, since only a limited number of nodes will send their reply to a multicast request.

This scheme distributes node replies over time and alleviates potential congestion in the domain. To the best of our knowledge this scenario has not been considered in the earlier work. We model and analyze the IoT domain performance under variable distributions of lifetime, size of stale record group, and number of nodes in the domain.

The paper is organized as follows. Section 2 gives basic notions about CoAP and multicasting, while Section 3 describes in detail the operation of the multicasting proxy at the IoT gateway. In Section 4 we present kernel based technique for data lifetime estimation and use it to estimate data lifetime and analyze the stale time of a record in Section 5, followed by the estimation of the period between multicasting requests in Section 6. Section 7 presents the results of performance evaluation within the IoT domain, preceded by a short discussion of communications in IoT domain. Finally, Section 8 concludes the paper.

2 CoAP and Multicast CoAP

Constrained Application Protocol (CoAP) is a lightweight replacement for HTTP that follows the

Representational State Transfer (REST) paradigm [2] and implements a subset of HTTP methods, namely GET, PUT, POST and DELETE. CoAP is specifically tailored to support machine to machine communications (M2M) in IoT networks by providing resource discovery, asynchronous communications, multicasting and observation streams of data [15]. Unlike HTTP which operates using TCP as its transport protocol, CoAP uses UDP for efficiency reasons: UDP provides basic functions such as protocol port multiplexing and error checking, contrary to TCP which has built-in reliability, congestion control and flow control. Congestion and flow control are not (yet) the paramount issues in IoT due to comparatively small traffic volume, but reliability is a crucial property that could not be left to the applications – it had to be provided within CoAP itself. Therefore CoAP architecture may be considered to have two layers: the upper layer deals with requests and responses through a number of methods, while the lower one implements reliable message transfer over UDP.

To this end, the messaging layer labels each message as confirmable (CON) or non-confirmable (NON). CON messages must be acknowledged with an acknowledgement message (ACK). Unrecognized CON messages will provoke the recipient to send a reset message (RST). Reliability is achieved using a simple stop-and-wait protocol with exponential backoff mechanism that will repeat the message in case an ACK message is not received within the predefined timeout. CON message and its ACK are related through the message ID which is part of the regular message header.

The upper, method layer uses services of the messaging layer to transmit CoAP methods and responses. In the simplest case, request will be sent as a CON message and response will be piggybacked onto the ACK message. However, it is possible that the IoT server node cannot provide the response as part of the acknowledgement. In such cases, response will be sent later in a separate CON message which needs to be acknowledged by the recipient (client or proxy). As the message ID of the response message is different from that of the CON request, the asynchronous request and response are linked through the value of a token that was provided in the header of the initial request message. A token is valid until the response is received, or its lifetime (the default value of which is 250 seconds) expires.

Requests and responses can also be sent as non-confirmable messages, again linked by the same token value. This approach makes perfect sense if there are few hops between the message source and recipient, and the underlying data link layer supports reliable messaging. For example, in IEEE 802.15.4 and IEEE 802.11ah standards, in acknowledgement mode, a packet can be re-transmitted several times before failure is declared [9].

2.1 On Proxy Operation and Multicasting

CoAP encourages proxying with caching since this reduces network traffic, facilitates access to devices in power saving mode (sleep) and provides resource hiding towards the rest of Internet which improves security. Moreover, it is simpler and more secure if the IoT node performs (mutual) authentication with the proxy than with an arbitrary client.

To reduce bandwidth usage and energy consumption of IoT server nodes which are, more often than not, battery operated, CoAP provides two special features. One of these is the observation mode [3] which is essentially a publish-subscribe relationship in which the proxy subscribes to notifications from a single IoT server node using a GET method with the appropriate option (`-observe`). The server's positive response indicates a promise to send notifications whenever the value of the observed physical variable changes. The proxy may request the data repeatedly; the server can send the data more often or even periodically. Either side may terminate the relationship, but the proxy must occasionally confirm its continuing interest in receiving the notifications. Note that this feature enables the reverse-mode operation of the proxy described above.

The other special feature is multicasting [12, 15] which may be considered as a CoAP extension that aims to improve scalability and efficiency. Multicast can be run in link-local mode over the IoT domain reachable over single hop, or over multiple hops beyond link-local scope provided forwarding nodes are capable of multicast routing.

Multicast request is sent as a multicast GET (mGET) method addressed to the IP multicast address and it is carried in a non-confirmable message. Servers reply in non-confirmable unicast with data, in which case the response token must match the request token. If the requested resource is not found, the server may reply accordingly or simply withhold the response [12].

Application based multicast groups can be pre-configured or created and deleted dynamically. Dynamic creation of groups can be done by special configuration node (e.g proxy). Information about group membership in link-local environment can be created/updated/deleted by special node (proxy) and communicated to the node(s) using PUT method. Addition of new multicast group for the node(s) can be done using POST method. Differentiation among multicast groups can be done by different IPv4/IPv6 multicast addresses or by UDP protocol ports different from the default one (5683).

Forming multicast groups is crucial for further multicast operation. Membership of multicast groups is created according to the application needs, to perform resource discovery, or to access physical variables sensed in locations defined with Universal Resource Identifier (URI). Clients can have a pre-configured list of groups or they can learn about multicast groups using service discovery, either using the default multicast group 'All CoAP nodes' [1] or via DNS-based service discovery at default UDP protocol port 5863 [11, 12, 13, 14]. Servers that have joined the multicast group then respond with links that correspond to entry points to resource interfaces they host. Resource discovery needs to be repeated periodically because nodes may join or leave the multicast group. Core link format described in [11, 14] also supports query filtering but we do not model this feature in the present work.

Token management is an important issue when multicasting is used. Namely, in unicast mode, reception of reply from IoT node frees the token value, but in multicast it can be released when 'most' of the replies have been received. Also, the token lifetime limits the number of tokens that can be kept 'alive' at any given time.

3 Multicasting Proxy with Data Lifetime Estimation

We assume that nodes in the IoT domain estimate homogeneous variables and that the proxy has an estimate of data freshness, initially obtained when the nodes (some of them, at least) begin by using POST method to update the corresponding records in the cache. In case IoT nodes report readings of heterogeneous variables or several distinct physical variables (e.g., temperature and humidity), the model has to be extended with multiple traffic classes, but the operation of the model would remain the same for each traffic class.

3.1 Proxy Operation

Operation of a multicasting proxy with estimation of data lifetime is schematically shown in Fig. 1 for the IoT network with five nodes. Circles with inscribed numbers correspond to important events, as described below.

The proxy periodically checks the freshness of cache data against the current freshness estimate. Event 1 corresponds to the moment when the number of stale records, i.e., those for which the lifetime has exceeded the current estimate, reaches a predefined threshold θ (equal to 2 in the example shown in the figure). Then, the proxy issues a multicast GET request (mGET) to the local IoT domain to refresh the data (event 2) which all nodes receive.

However, nodes do not respond to it immediately. Instead, each node waits for a randomly chosen leisure time period L [15] and then checks the physical variable of interest. If its value has changed from the last reading, the node responds to the mGET request with a '2.05 content' message containing newly read data (event 3). Upon receiving this value, the proxy updates the cached record and resets the freshness timer (event 4). The cached data is immediately available to respond to client requests.

The leisure time spreads node responses over time within a predefined leisure time window (to be explained in detail below) which reduces the contention and the resulting transmission failures in the wireless IoT domain [15]. The original documentation actually asks for data reading to be performed prior to starting the leisure interval timer countdown. However, switching the order of those actions ensures that the data transmitted is more fresh, as it may change its value during the leisure countdown. Furthermore, the wait-then-sense ordering does not affect the latency of data received at the proxy as it receives the data only after the leisure period in both scenarios.

It is also possible that the value of the physical variable has not changed since the last reading (event 5). In this case, the IoT server simply ignores the mGET request and remains silent. This reduces the volume of traffic in the wireless domain and energy expenditure of server nodes.

This procedure is repeated every time the proxy notes that the number of stale records reaches the threshold θ, as shown in the second and third mGET cycle in Fig. 1. Each multicast request actually presents a different token value to with the objective of synchronizing and refreshing data values at the cache.

Sometimes a node will record and subsequently transmit a new data value even if the freshness period of the cached data value has not exceeded the lifetime estimate (event 6). In this case, the proxy simply overwrites the cached data value and resets it lifetime estimate.

In all cases, nodes that reply to a mGET request label their responses using the token of the most recent mGET request. As the result, node reports can be grouped by node IP address and, within the group, by tokens. Token value can be numbered sequentially or as a function of time to further simplify data processing. This approach introduces virtual partitioning of the multicast group with evenly distributed replies, compared to the traditional approach in which every node in the domain had to reply to a mGET request [8].

3.2 Further Observations

It is worth noting that the technique described above is somewhat similar to observation mode but with an

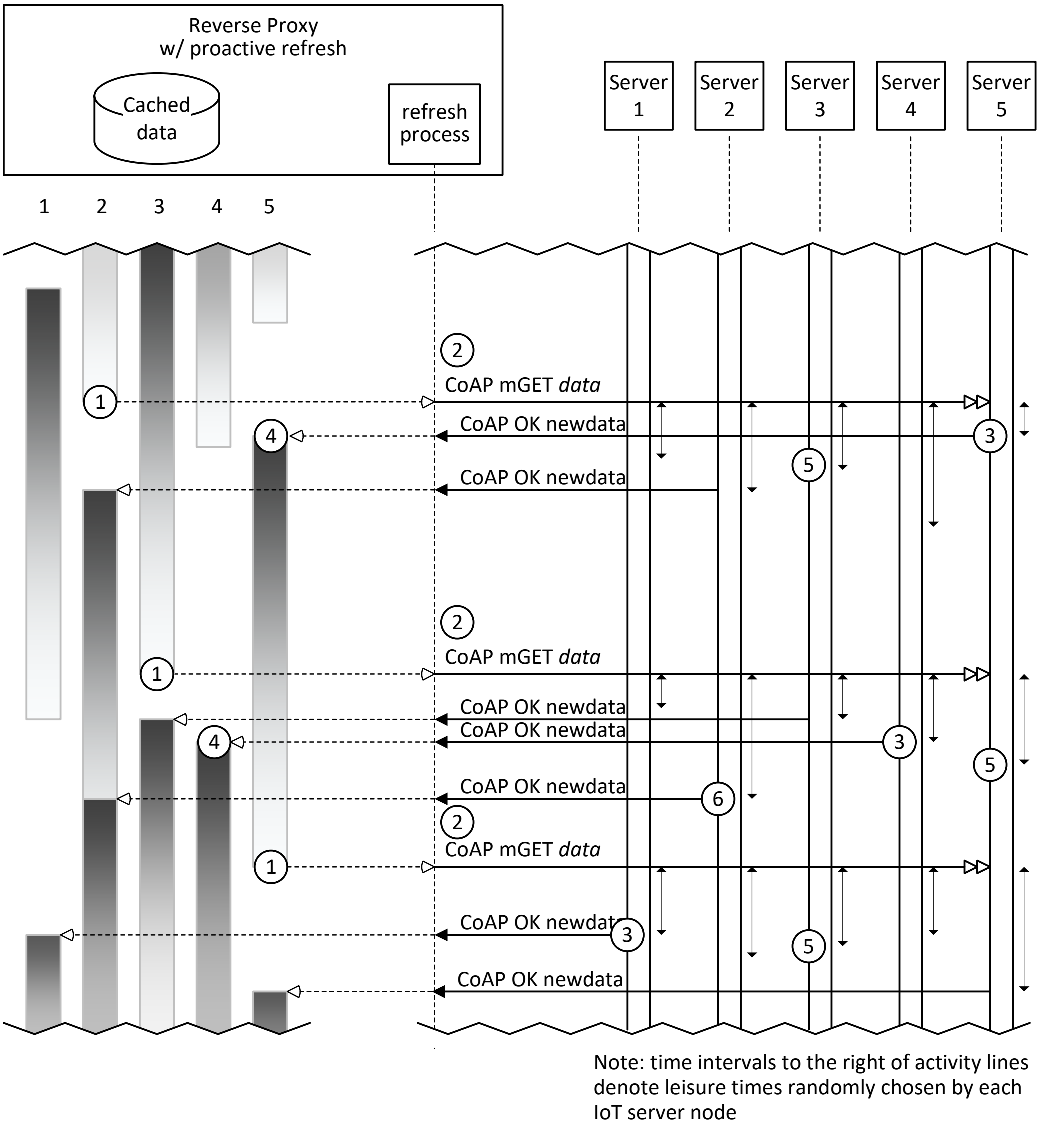


Figure 1: Pertaining to the operation of multicast proxy

important difference. Regular observation mode is a relationship between the proxy and a single IoT server node; in our approach, the entire multicast group effectively participates in the observation. Compared to regular CoAP multicast, this scheme actually reduces the volume of traffic in the IoT domain since only nodes which obtain new data values between two consecutive mGET requests with tokens t_n and t_{n+1} will reply to the mGET request with token t_n. This may be considered as a form of reply suppression which was recommended in [12] but without directions how to implement it in practice. The reply would be sent in a non-confirmable message with reply code '2.05 content' and the new data value; any other outcome will be suppressed by IoT server.

3.3 Protocol Design

Regarding protocol design of the proxy, we assume that the protocol stack consists of application, transport and network and data link layers, with the last one implemented on the networking adapter. Similar layered architectures were proposed in [6, 18]. Fig. 2 shows the stage that corresponds to proactive cache operation.

Network and protocol layers are each served with one upwards and one downwards thread. Application layer has multiple concurrent threads for record reading and one for record updating. Although time intervals between mGET requests do not follow exponential distribution, due to the low request rate and combination with replies to POST methods we can approximate downlink traffic as Poisson. Traffic in the uplink direction can also be approximated as Poisson due to the presence of leisure time in replies and existence of messages with POST methods. Queuing analysis of these three layers is presented in detail in [8] and it will not be repeated here.

4 Kernel-Based Estimation of Random Variables

The proxy performs record lifetime estimation using kernel based technique [17]. To achieve reliable estimation of probability distribution function (pdf) of record lifetime, the proxy needs to collect a large number (of the order of several hundreds) of inter-POST periods and find optimal bandwidth for estimation. In this work we assume that record lifetime is subexponential, more precisely we assume that it follows family of Erlang-k probability distributions [5]. We consider this approach to be more realistic compared to the exponential distribution which we used in our previous work [8, 10]. So far many kernel functions can be used such as Epanechnikov, Biweight, Triweight, Gaussian, triangular and uniform [17]. In this work we use Gaussian kernel $G(t) = \frac{1}{\sqrt{2\pi}}e^{-\frac{t^2}{2}}$ due to ease of analytical manipulations.

Gaussian kernel has the variance and integral of its square value, respectively, given by

$$\mu_2(G) = \int_{-\infty}^{\infty} t^2 G(t) dt = 1 \tag{1}$$

$$R(G) = \int_{-\infty}^{\infty} G(t)^2 dt = \frac{1}{2\sqrt{\pi}} \tag{2}$$

Expression (1) has the variance and standard deviation of one and in practical cases it has to be scaled around measurement points according to the inter-point distance. Therefore we need to work with the gaussian kernel which has variance equal to b^2 as:

$$G_b(t) = \frac{1}{b\sqrt{2\pi}} e^{-\frac{t^2}{2b^2}} = \frac{1}{b} G\left(\frac{t}{b}\right) \tag{3}$$

with $\mu_2(G_b) = b^2$ and $R(G_b) = \frac{1}{2b\sqrt{\pi}}$ The standard deviation value b is also known as the estimation bandwidth and it is very important for the quality of estimation. With the known bandwidth and in the presence of n measurements, probability distribution of random variable under observation can be obtained as the sum of kernels positioned around the measurement data. For example, if we are estimating probability distribution of data lifetime at the proxy and we have collected n measurements of data lifetime denoted as $T_{l,i}$, $i = 1 \ldots n$, its pdf can be estimated by

$$\begin{aligned} \hat{T}_l(t) &= \frac{1}{n} \sum_{i=1}^{n} G_b(t - T_{l,i}) \\ &= \frac{1}{nb\sqrt{2\pi}} \sum_{i=1}^{n} e^{-\frac{(t-T_{l,i})^2}{2b^2}} \end{aligned} \tag{4}$$

Quality of estimation is evaluated in two steps. First the mean squared error (MSE) related to single measurement point is calculated:

$$\begin{aligned} MSE(\hat{T}_l) &= E(\hat{T}_l(t) - T_l(t))^2 \\ &= (E\hat{T}_l(t) - T_l(t))^2 + E(\hat{T}_l(t) - E\hat{T}_l(t))^2 \\ &= Bias(\hat{T}_l(t))^2 + Var(\hat{T}_l(t)) \end{aligned} \tag{5}$$

After that mean integrated squared error (MISE) is computed based on whole measurement space:

$$\begin{aligned} MISE(T_l\hat{(t)}) &= \int_{-\infty}^{\infty} MSE(\hat{T}_l(t)) dt \qquad (6) \\ &= \int_{-\infty}^{\infty} Bias(\hat{T}_l(t))^2 dt + \int_{-\infty}^{\infty} Var(\hat{T}_l(t)) dt \end{aligned}$$

However, finding appropriate bandwidth requires further calculations. Optimal bandwidth is determined by minimizing mean integrated squared error (MISE). In order to find minimum in a computationally feasible way, expression (6) for MISE has to be simplified by performing a Taylor series expansion from which only the most important members are retained:

$$MISE(T_l\hat{(t)}) \approx \frac{1}{4} b^4 \mu_2(G)^2 R(T_l(t)'') + \frac{R(G)}{bn} \tag{7}$$

which renders itself to derivation of bandwidth that gives the minimal MISE as

$$b_{opt} = \left(\frac{R(G)}{n\mu_2(G)^2 R(T_l'')^2}\right)^{1/5} \tag{8}$$

In case of Gaussian kernels previous expression becomes:

$$b_{opt,g} = \left(\frac{1}{2\sqrt{\pi} n R(T_l'')^2}\right)^{1/5} \tag{9}$$

Optimal bandwidth for data that is distributed close to the normal distribution is obtained by further substituting $R(T_l'')$ for normal distribution:

$$b_{ns,g} = \hat{\sigma} \left(\frac{8\sqrt{\pi} R(G)}{3\mu_2(G)^2 n}\right)^{1/5} = \hat{\sigma} \left(\frac{4}{3n}\right)^{1/5} \tag{10}$$

where for large number of data points $\hat{\sigma}$ denotes standard deviation. If the data sample is not large

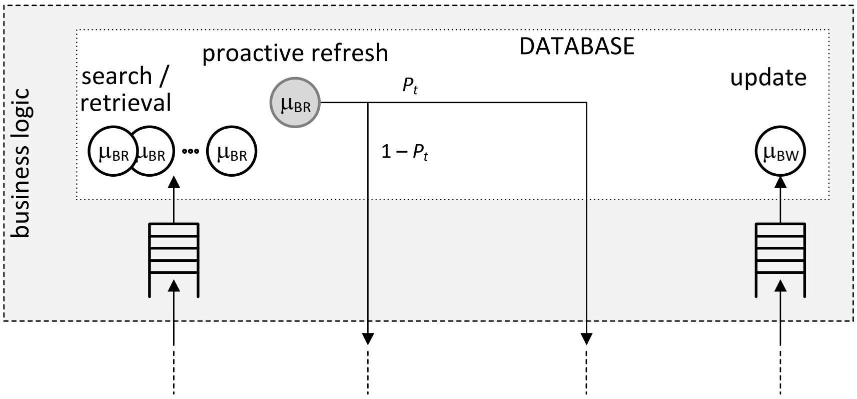


Figure 2: Queuing stages of the proxy with proactive cache refresh

then smaller between standard deviation of data sample $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(T_{l,i} - \overline{T_l})$ and scaled interquartile range R. Interquartile range contains 50% of data samples. If interquartile range is used instead of standard deviation then data outliers can not significantly affect the optimal estimation bandwidth. Therefore we used $\hat{\sigma} = min(R/1.34, s)$.

For example for Erlang-k distribution using Gaussian kernels optimal bandwidth can also be calculated by computing $R(f'')$ and plugging it in (9).

However, if the original probability distribution is not known, value $R(T_l'')^2$ in (8) is not known either and further estimation is needed. First step in this estimation is

$$\hat{T_l}''(t) = \frac{1}{b^3 n}\sum_{i=1}^{n} G''\left(\frac{t - T_{l,i}}{b}\right) \tag{11}$$

Then we apply biased cross validation (BCV) to find $R(\hat{T_l}'')$ [17], based on computing convolution between the two kernels, as

$$\begin{aligned} R(\widetilde{T_l}''(\Delta)) &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\int_{w=-\infty}^{\infty} G_b''(\Delta_{i,j} - w)G_b''(w)dw \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n}(G_b'' * G_b'')(\Delta_{i,j}) \end{aligned} \tag{12}$$

Calculating the minimal value of MISE requires iterative computation of expression (12) for a range of bandwidth values (starting with well known oversmoothed bandwidth [17]) and computation of MISE as

$$MISE_{BCV}(b) = \frac{1}{4}b^4\mu_2(G)^2R(\widetilde{T_l}'') + \frac{R(G)}{bn} \tag{13}$$

and the bandwidth which results in smallest MISE is selected for estimation as b_{BCV}.

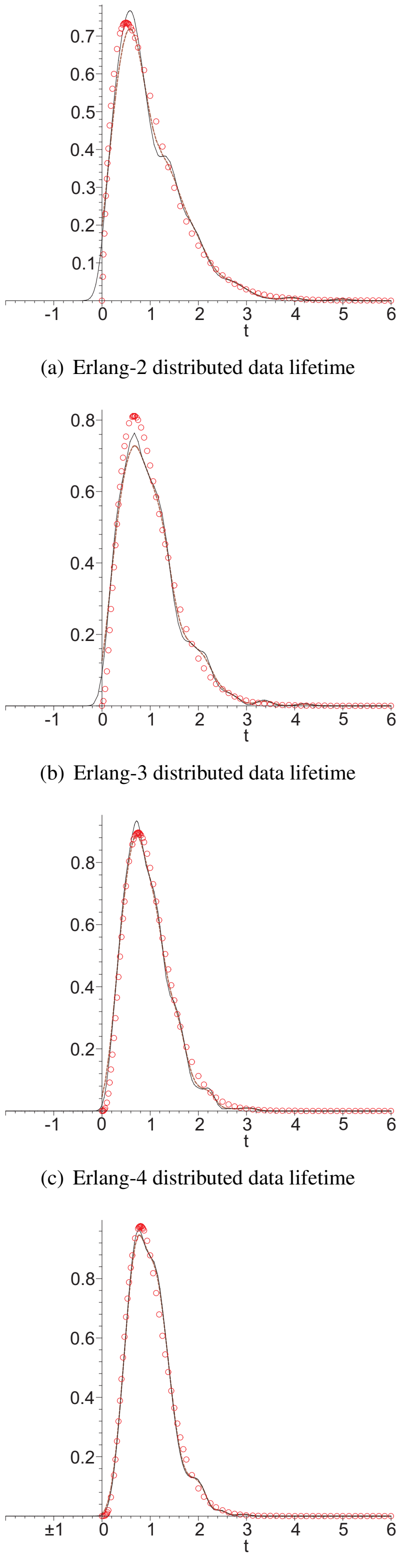
Table 1: Approximations of optimal bandwidth

Distribution	$b_{ns,g}$	$b_{erlangK,g}$	b_{BCV}
Erlang-2	0.193	0.107	0.134
Erlang-3	0.172	0.104	0.124
Erlang-4	0.15	0.112	0.119
Erlang-5	0.132	0.109	0.119

5 Estimation of Record Lifetime

In our experiments we have considered Erlang-k, $k = 2..5$, distributions of data lifetime with mean value of $\overline{T_l} = 1$ minute (60s). Data samples of lifetime were obtained using Maple's Statistics package. Estimation was done on a sample of 500 measurements.

Fig. 3 shows the estimation of pdf for the data lifetime where red circles present mathematical expression, black line shows estimation with optimal bandwidth derived from BCV algorithm, and brown dashed line presents the estimate obtained using bandwidth $b_{ns,g}$ appropriate for normal distribution [17]. The optimal bandwidth provides best estimation but estimation with $b_{ns,g}$ (when $k > 2$) gives results close to optimal and leads to lightweight lifetime distribution estimation. Table 1 presents values of bandwidth corresponding to normal and Erlang-k distributions respectively as well as optimal bandwidth calculated using BCV approach. The reason why the values become closer as the degree of distribution increases lies in the fact that Erlang-k distributions with increasing value of k have smaller coefficients of skewness and kurtosis which means that they are reasonably symmetrical around the mean and have fast decaying tails. From this estimation we can compute higher moments of data lifetime and use them to determine the threshold T_{hr} after which the record



(a) Erlang-2 distributed data lifetime

(b) Erlang-3 distributed data lifetime

(c) Erlang-4 distributed data lifetime

(d) Erlang-5 distributed data lifetime

Figure 3: Estimation of pdf for record lifetime

is considered as outdated. Mean value and standard deviation can be found as

$$\overline{T_l} = \int_0^\infty t\hat{T_l}(t)dt = \int_0^\infty \frac{1}{n}\sum_{i=1}^{n} tG_b(t-T_{l,i}) \tag{14}$$

$$\sigma(T_l) = \left(\int_0^\infty \frac{1}{n}\sum_{i=1}^{n}(t-\overline{T_l})^2 G_b(t-T_{l,i})\right)^{1/2} \tag{15}$$

As explained in Section 3 above, we consider the refreshment policy where the proxy checks for θ outdated records and then sends a multicast GET request. Fig. 4 shows the details of data lifetime expiration between two subsequent mGET requests for a cache with a limit of $\theta = 2$ records that can become stale before issuing the next mGET request. From Fig. 4 we can make the following observations:

- Moment of generation of data R_{i+1} is a random point in lifetime of data R_i. Distance between two successive data generation points R_i and R_{i+1} from the point of view of renewal theory [4] is known as elapsed data lifetime with probability distribution $T_{l,-}(y)$.
- Given that the threshold value is changing slowly, i.e., it is constant over a period of time which is large compared to the threshold itself, the distances between two consecutive moments when value of freshness threshold is exceeded have same probability distribution of elapsed data lifetime $T_{l,-}(y)$.

The value of data freshness threshold should be chosen to satisfy two goals. First, it should be large enough to prevent too frequent refresh requests. Second, it should be low enough so that the group refresh policy described below triggers a refresh request while most of the data records are still sufficiently fresh so that client data requests are still served with valid data. In our work, we calculate the threshold value using the mean and standard deviation of estimated data lifetime as

$$T_{hr} = m\overline{T_l} + l\sigma(T_l) \tag{16}$$

where $0 < m \leq 1$ and $0 < l \leq 3$.

Probability density function (pdf) of elapsed data lifetime [16] can be computed as

$$T_{l,-}(y) = \frac{1}{\overline{T_l}}\int_{t=y}^{\infty} \hat{T_l}(t)dt \tag{17}$$

with mean value and standard deviation obtained as

$$\overline{T_{l,-}} = \int_{y=0}^{\infty} yT_{l,-}(y)dy \tag{18}$$

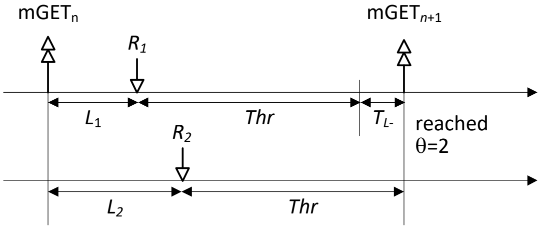


Figure 4: Timing of refresh requests and responses for threshold of $\theta = 2$ records

and

$$\sigma(T_{l,-}) = \int_{y=0}^{\infty} (y - \overline{T_{l,-}})^2 T_{l,-}(y) dy, \qquad (19)$$

respectively. The total (threshold based) stale time for θ records is the sum of $\theta - 1$ elapsed lifetimes, with mean value of $(\theta - 1)\overline{T_{l,-}}$.

We also need to include the impact of the leisure period. As discussed in Section 3, each node will start a leisure period countdown after the mGET request; if fresh data is present at the time the countdown ends, it will be transmitted to the proxy. We assumed that the window for leisure period takes a fixed portion $W_L = (\theta - 2)\overline{T_{l,-}}$ of the period between multicast requests, so that the ratio $W_L/T_{MGET} = (\theta - 2)/\theta$. The reply window period W_L is divided in slots of duration t_{boff} which match the backoff period of the underlying medium access control (MAC) protocol (see next Section), so the number of slots is $n_s = \frac{W_L}{t_{boff}}$. Each node selects its slot – effectively, the leisure time period L – following a uniform distribution with probability generation function (PGF) of

$$N(z) = \sum_{i=0}^{n_s - 1} \frac{1}{n_s} (z)^i \qquad (20)$$

and mean leisure time becomes $\overline{L} = t_{boff}\overline{N}$.

Since leisure time is much longer than the backoff at the MAC layer, the probability that i-th record in group of θ records is stale can be obtained as

$$P_{st}(i) = \int_{t=0}^{T_{hr}+(i-1)\overline{T_{l,-}}+\overline{L}} \hat{T_l}(t) dt \qquad (21)$$

and find mean probability that any record is stale, i.e., that it exceeds the threshold value, as

$$P_{ast} = \frac{1}{\theta} \sum_{i=1}^{\theta} P_{st}(i). \qquad (22)$$

Fig. 5 shows mean value of the total estimated stale time, expressed in minutes, and its coefficient of variation and probability that any record is stale when $T_{hr} = \overline{T_l}$. Parameter of Erlangian distribution for lifetime was varied between 1 and 6, and record group size was varied from 2 to 10. We notice that mean record stale time increases with the record group size which is expected. However it decreases with the parameter of Erlangian distribution since the elapsed record lifetime decreases with the distribution parameter. Coefficient of variation of total stale time decreases both with the increase of record group size and parameter of Erlangian distribution of lifetime. Finally, the probability that any record is stale (according to the threshold) shows a mild decrease when Erlangian parameter is increasing but a rapid increase with increase of group size.

6 Estimation of Rate of Multicast GET Requests

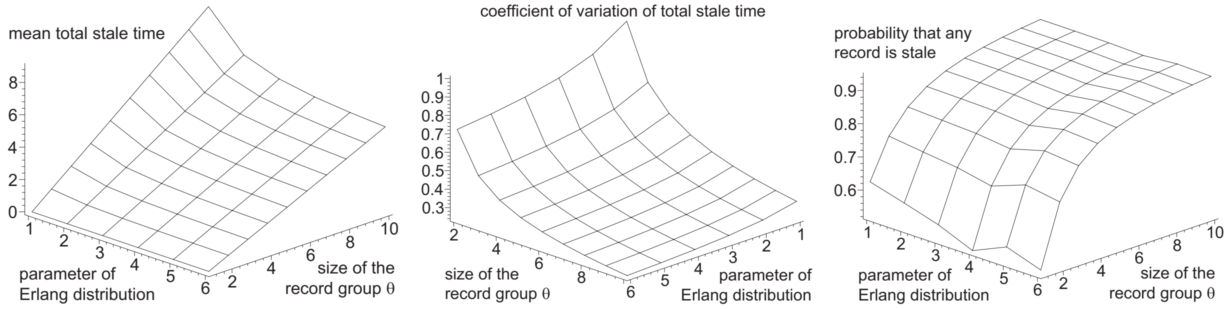
Let us now we look into the situation when groups of θ records become stale in sequence, and thus result in a sequence of mGET requests. Clearly the time period between two consecutive mGET requests is the sum of θ elapsed data lifetimes. Probability distribution of this time can be found as a θ-fold convolution of elapsed data lifetime. Since data records are homogeneous, elapsed data lifetimes are i.i.d. variables, and mean and variance of the sum is equal to the sum of means and variances respectively. Therefore mean period between multicast get requests and rate of requests are

$$T_{MGET} = \theta\overline{T_{l,-}} \qquad (23)$$

$$\lambda_{mGET} = \frac{1}{T_{MGET}} \qquad (24)$$

Although all nodes in the IoT domain served by the proxy belong to the same multicast group, only nodes where the physical variable has changed the value between the previous and current multicast will send replies. Given the randomness of data lifetime and imperfection of estimation, it is possible that more than θ nodes will reply but that number will still be considerably smaller than the number of nodes in the IoT domain.

Possible transmission collisions from IoT nodes (besides the addition of a random leisure period before each transmission) are further resolved using the contention resolution mechanism(s) of the underlying MAC protocol. This approach ensures that all responses arrive at the proxy within a time period equal to the sum of leisure period and maximum backoff period of the MAC algorithm. Therefore, under unsaturated MAC operation, it is almost impossible for a reply to a multicast request with token t_n to be delayed for so long that it eventually arrives during the reply period of next



(a) Mean total stale time

(b) Coefficient of variation of total stale time

(c) Probability that any record in the group is stale according to the threshold

Figure 5: Descriptors of total record stale time

request with t_{n+1}. This means that the token t_n can be safely released when a new request is issued.

We will now show how to approach modeling of uplink and downlink packet rates, assuming there are n_d nodes in the IoT domain. We note that estimation of data lifetime distribution at the proxy requires POST methods from small number, e.g., 10% of nodes. Reply to the POST method is CoAP reply '2.01 created'. The downlink rate from proxy application layer, through protocol and network layer, to wireless medium is only

$$\lambda_{dsp} = \lambda_{MGET} + 0.1 n_d \lambda_l \tag{25}$$

where $\lambda_l = \frac{1}{\overline{T_l}}$ is mean data update rate. In the uplink direction, there are estimation POST messages and replies to multicast request which is non-confirmable as are the replies. (Lack of reliability is not a big problem since the MAC layer supports it through retransmissions.) Their total rate is

$$\lambda_{usp} = n\lambda_{MGET} + 0.1 n_d \lambda_l. \tag{26}$$

These values can be used to find upstream and downstream delays, respectively, in IoT proxy stages.

The downlink rate towards the IoT node contains only multicast GET methods and replies to POST methods, while the outgoing rate from the IoT node is comprised of replies to mGET requests and estimation POST methods. Then the rate from IoT node towards the proxy is

$$\lambda_{tot} = \lambda_{MGET} + 0.1\lambda_l. \tag{27}$$

7 Performance Evaluation

7.1 Model of Communications

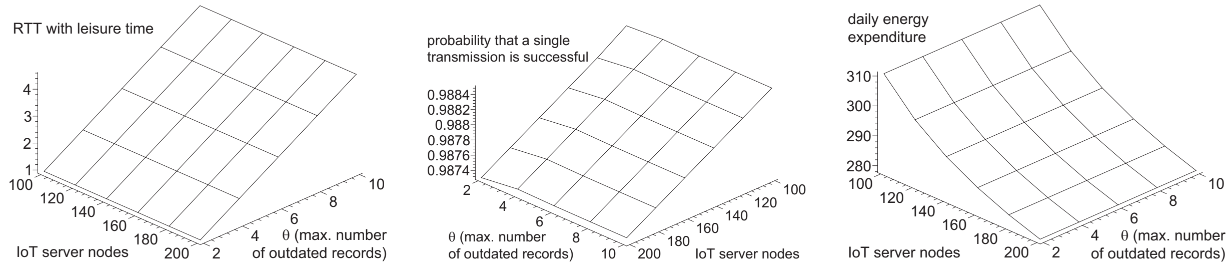
We consider a single-hop 6LoWPAN over IEEE 802.15.4 cluster with acknowledged transfer. The cluster operates in the ISM band at 2.4GHz with raw data rate 250kbps, set up so that period between the beacons is *BI* $= 0.98$s. Thus, IoT nodes can wake up every 0.98s to hear the information from the beacon which contains list of nodes with pending downlink traffic. Backoff parameters such as the minimum backoff exponent, the maximum value of the backoff exponent and the maximum number of backoff attempts are set to their default values of *macMinBE*=3, *aMaxBE*=5 and *macMaxCSMABackoffs* = 4, respectively, as specified in [9]. Thus we can assume that in the network with a reasonably low bit error rate of $BER = 10^{-5}$ and nodes that send a single CoAP message every 1min (60 seconds) on the average, four retransmission attempts should provide sufficient reliability to compensate for the lack of CoAP reliability due to nonconfirmable messages. Packet size is 127 bytes, including 6LoWPAN, UDP and CoAP headers.

The operation of the MAC layer in this setting can be evaluated using an analytical model, but we do not present it here as its detailed derivation is available in [9] and its application to multicasting proxy is presented in [8].

7.2 Performance Results

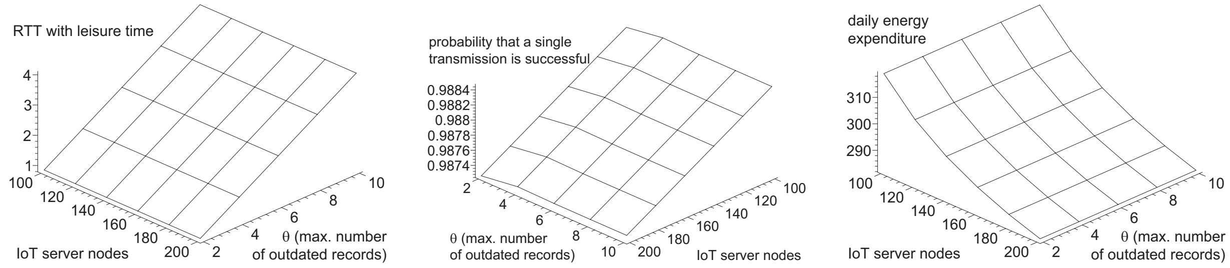
We have evaluated the performance of multicasting proxy with three distributions of data lifetime, namely with Erlang-2, -3, and -4 distributions with the same mean value of $\overline{T_l} = 60$s. Proxy used proactive refreshing of data with multicasting requests. The number of stages at proxy was three [6], and mean stage execution time was set to $\frac{1}{\mu} = 5$ms.

IoT proxy sends a mGET request when $\theta = 2\,..\,10)$ data records become stale. Threshold for data freshness is set to $T_{hr} = \overline{T_l} = 60$s. The number of IoT devices was varied between 100 to 200. We assumed bit error rate of 10^{-5} and 6LoWPAN packet size of 127 bytes. We did not use any explicit leisure time since its role has been



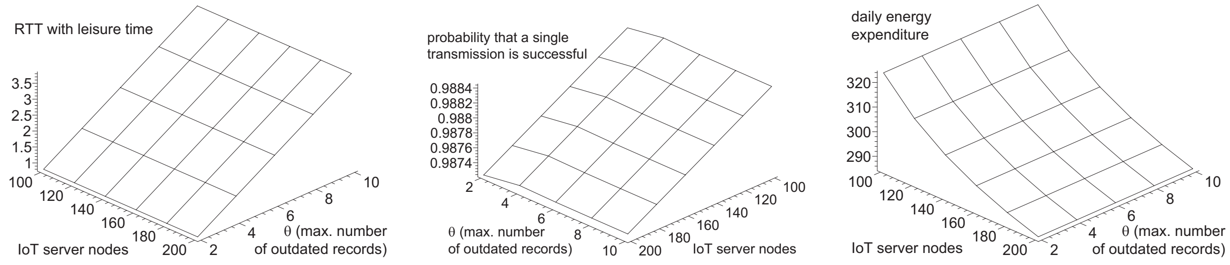
(a) Round trip time (in seconds) (b) Transmission success probability (c) Mean daily energy expenditure (in Joules) by the IoT node

Figure 6: Proxy performance with Erlang-2 distributed data lifetime for uniformly generated leisure time within the leisure period



(a) Round trip time (in seconds) (b) Transmission success probability (c) Mean daily energy expenditure (in Joules) by the IoT node

Figure 7: Proxy performance with Erlang-3 distributed data lifetime for uniformly generated leisure time within the leisure period



(a) Round trip time (in seconds) (b) Transmission success probability (c) Mean daily energy expenditure (in Joules) by the IoT node

Figure 8: Proxy performance with Erlang-4 distributed data lifetime for uniformly generated leisure time within the leisure period

fulfilled by the elapsed data lifetime.

We present round trip time, probability of successful transmission and daily energy consumption per sensor for three probability distributions of data lifetime namely Erlang-2, -3, and -4. Results are shown in Figs. 6, 7 and 8, respectively. We observe that the round trip time with included leisure period decreases when the parameter of Erlangian distribution increases. This is a consequence of decreased period between mGET requests due to the decrease of elapsed lifetime value. Related result is increase of daily energy consumption per device when the parameter of Erlangian distribution is increasing since mGET request rate is increasing. Energy expenditure decreases with the increase of the number of IoT devices since one downlink mGET transmission serves increasing number of sensors.

Probability of successful transmission slightly decreases when the parameter of Erlangian distribution is increasing due to the increase in multicast request rate. Due to low traffic in the IoT domain, the diagrams presented do not show significant dependence on the number of IoT devices which also means that this technique can be applied to domains with 300-400 IoT devices without severe performance deterioration.

These results show that the probability distribution of data lifetime has a significant impact on the proxy management scheme. Data lifetime distributions with small values of standard deviation and skewness require more frequent cache updates for the same record group size. This increases energy consumption in IoT domain. If we combine these results with results from Fig. 5, we see that the record size for multicast refreshment should be determined from the limit on total estimated stale time of data records for known data lifetime distribution. Alternatively, instead of record size, the value of freshness threshold can be manipulated. If there is additional restriction on energy consumption per IoT node then limit on the number of IoT devices should be imposed.

8 Conclusion

In this paper we have developed the model of multicasting based proactive proxy which estimates data lifetime and uses that information to determine moments to refresh the cache. Cache refreshment decision is based on a number of stale data records. Results show that a comparatively small number of lifetime measurements, say, 500 or so, are sufficient for reliable estimation. Also, the bandwidth appropriate for the estimation of normal distribution resulted in estimates that were close to the optimal result from biased cross validation approach. In our model we have considered IoT domain consisting of a single hop 6LoWPAN/IEEE 802.15.4 cluster. Our findings show that if the standard deviation and skewness of the data lifetime distribution decrease for the same mean value of the variable and same record size, the period of multicast requests decreases. This leads to increased traffic load and, perhaps more importantly, to increased energy consumption in the IoT domain. This situation can be remedied by decreasing the number of sensors in the domain or by increasing the data freshness threshold.

References

[1] C. Bormann and Z. Shelby, “RFC 7959: Block-wise transfer in the constrained application protocol,” https://tools.ietf.org/html/rfc7959, 2016.

[2] R. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, Irvine, CA, 2000.

[3] K. Hartke, “RFC 7641: Observing resources in the constrained application protocol,” https://tools.ietf.org/html/rfc7641, 2015.

[4] D. P. Heyman and M. J. Sobel, *Stochastic models in operations research*. New York: McGraw-Hill, 1982, vol. I: Stochastic processes and operating characteristics.

[5] L. J. Kleinrock, *Queuing systems*. New York: John Wiley and Sons, 1972, vol. I: Theory.

[6] F. M. Kovatch, “Scalable web technology for the internet of things,” Ph.D. dissertation, ETH Zürich, Switzerland, 2015.

[7] J. Mišić, M. Z. Ali, and V. B. Mišić, “Protocol architectures for CoAP IoT domains,” *IEEE Network*, vol. 32, no. 4, pp. 811–817, 2018.

[8] J. Mišić and V. Mišić, “Proxy cache maintenance using multicasting in CoAP IoT domains,” *IEEE Internet of Things Journal*, 2018.

[9] J. Mišić and V. B. Mišić, *Wireless Personal Area Networks: Performance, Interconnections and Security with IEEE 802.15.4*. Chichester, UK: John Wiley & Sons, Jan. 2008.

[10] J. Mišić, V. B. Mišić, and X. Chang, “Kernel based estimation of domain parameters at IoT proxy,” in *IEEE Globecom*, Abu Dhabi, UAE, Dec. 2018.

[11] M. Nottingham and S. Hammer-Lahav, “RFC 5785: Defining well-known uniform resource identifiers,” https://tools.ietf.org/html/rfc5785, 2010.

[12] A. Rahman and E. Dijk, “RFC 7390: Group communications for the constrained application protocol,” https://tools.ietf.org/html/rfc7390, 2014.

[13] C. Shelby, Z. Bormann and S. Krco, “Core resource directory,” https://tools.ietf.org/html/draft-ietf-core-resource-directory-09, 2016.

[14] Z. Shelby, “RFC 6690: Constrained RESTful environments (core) link format,” https://tools.ietf.org/html/rfc6690, 2012.

[15] Z. Shelby, “RFC 7252: The constrained application protocol,” https://tools.ietf.org/html/rfc7252, 2014.

[16] H. Takagi, *Queueing analysis*. Amsterdam, The Netherlands: North-Holland, 1991, vol. 1: Vacation and Priority Systems.

[17] M. Wand and M. Jones, *Kernel smoothing*. Chapman & Hall, Inc., 1995.

[18] M. Welsh, D. Culler, and E. Brewer, “SEDA: an architecture for well-conditioned, scalable internet services,” *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 230–243, 2001.

[19] D. Zordan, R. Parada, M. Rossi, and M. Zorzi, “Automatic rate-distortion classification for the IoT: Towards signal-adaptive network protocols,” in *IEEE GlobeCom*, Singapore, Dec. 2017.

AUTHOR BIOGRAPHIES

Jelena Mišić is Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. Her research interests include Internet of Things, blockchain-based systems, performance evaluation of wireless networks, and network security. She has published 4 books, over 125 papers in archival journals and over 200 papers at international conferences. She serves on editorial boards of *IEEE Transactions on Vehicular Technology*, *IEEE IoT Journal*, *IEEE Network*, *Computer Networks* and *Ad hoc Networks*. She is a Fellow of IEEE and Member of ACM.

Vojislav B. Mišić is Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. His research interests include Internet of Things, blockchain-based systems, performance evaluation of wireless networks and systems and software engineering. He serves on the editorial boards of *IEEE Transactions on Cloud Computing*, *Ad hoc Networks*, *Peer-to-Peer Networks and Applications*, and *International Journal of Parallel, Emergent and Distributed Systems*. He is a Senior Member of IEEE and member of ACM.

Xiaolin Chang is Professor at the School of Computer and Information Technology, Beijing Jiaotong University. Her current research interests include edge/cloud computing, network security, blockhain and blockchain-based systems, and security and privacy in machine learning.