Queueing Network Modeling Patterns for Reliable and Unreliable Publish/Subscribe Protocols

Georgios Bouloukakis, Ajay Kattepur, Nikolaos Georgantas, Valerie Issarny

Queueing Network Modeling Patterns for Reliable and Unreliable Publish/Subscribe Protocols

Georgios Bouloukakis[1,3] Ajay Kattepur[2] Nikolaos Georgantas[3] Valérie Issarny[3]
gboulouk@ics.uci.edu,ajay.kattepur@tcs.com,nikolaos.georgantas@inria.fr,valerie.issarny@inria.fr
[1]Donald Bren School of Information and Computer Sciences, University of California, Irvine, USA
[2]Embedded Systems & Robotics, TCS Research & Innovation, India
[3]MiMove Team, Inria Paris, France

ABSTRACT

Mobile Internet of Things (IoT) applications are typically deployed on resource-constrained devices with intermittent network connectivity. To support the deployment of such applications, the *Publish/Subscribe* (pub/sub) interaction paradigm is often employed, as it decouples mobile peers in time and space. Furthermore, pub/sub middleware protocols and APIs consider the Things' hardware limitations and support the development of effective applications by providing Quality of Service (QoS) features. These features aim to enable developers to tune an application by switching different levels of response times and delivery success rates. However, the profusion of pub/sub middleware protocols coupled with intermittent network connectivity result in non-trivial application tuning. In this paper, we model the performance of middleware protocols found in IoT, which are classified within the pub/sub interaction paradigm – both *reliable* and *unreliable* underlying network layers are considered. We model reliable and unreliable protocols, by considering QoS semantics for data validity, buffer capacities as well as the intermittent availability of peers. Finally, we perform statistical analysis by varying these QoS semantics, demonstrating their significant effect on the rate of successful interactions. We showcase the application of our analysis in concrete scenarios relating to *Traffic Information Management* systems, that integrate both reliable and unreliable participants. The consequent *PerfMP* performance modeling pattern may be tailored for a variety of deployments, in order to control fine-grained QoS policies.

KEYWORDS

Publish/Subscribe Middleware; Mobile Connectivity; Queueing Networks; Quality of Service.

ACM Reference Format:
Georgios Bouloukakis, Ajay Kattepur, Nikolaos Georgantas, Valérie Issarny. 2018. Queueing Network Modeling Patterns for Reliable and Unreliable Publish/Subscribe Protocols. In *Proceedings of ACM MobiQuitous conference (MobiQuitous'18).* ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.475/123_4

MobiQuitous'18, November 2018, New York, USA

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

1 INTRODUCTION

Internet of Things (IoT) applications have proliferated multiple sectors including the Industrial Internet, Smart Cities and Mobile Communication Systems [2]. While the sense–compute–actuate control flow functionality has received considerable attention [15], aspects such as resource limitations, intermittent network connectivity and suitable middleware paradigms for Quality of Service (QoS) dependent tasks are still under active study.

The underlying middleware of IoT applications tends to follow the Publish/Subscribe (pub/sub) interaction paradigm, that provides the advantages of space and time decoupling between peers [10]. Space decoupling implies that the peers are agnostic to each others addresses; time decoupling allows peers to be connected intermittently while still successfully receiving messages. This is particularly needed in the case of mobile IoT systems, where-in unreliability in the underlying network connectivity may be mitigated by message storage/retrieval from an intermediary broker. Multiple standards and toolkits based on the pub/sub paradigm have been proposed including MQTT [4], RabbitMQ [22] and Kafka [1].

As an illustration, consider popular traffic management systems [19, 28, 30, 31] that consists of: 1) *fixed-sensors* (vehicle detectors, traffic cameras, Doppler radars) that have been installed on existing infrastructure – they ensure the fastest but not reliable message delivery [15, 18] through the CoAP [25] "non-confirmable" delivery feature; 2) *vehicle-devices* (on-board) with GPS-based systems – they produce notifications via the MQTT [4] protocol using the "fire-and-forget" delivery feature; and, 3) *smartphones* with embedded sensors (accelerometer, gyroscope) that push/receive data via the reliable (built atop TCP) ZeroMQ [14] and AMQP [27] middleware protocols. The combination of such intelligent systems can provide us an overall *Traffic Information Management* (TIM) system in order to accurately estimate traffic conditions. For a TIM system to be used reliably by mobile applications, the *freshness* of procured data must be ensured. The TIM system guarantees the freshness of provided information by applying a `lifetime` period to each message. However, messages may be delivered with delay because of disconnections. A unified framework is needed to analyze delivery message success rates as well as end-to-end response time for message reception.

The focus of this paper is to analyze IoT applications' QoS using the pub/sub abstraction layer. As the pub/sub paradigm allows control of publisher/subscriber timeout periods and message lifetimes at the broker, an intricate model that captures such properties would result in improved reliability,

availability and performance of IoT applications. We model message passing and buffering at the application/middleware layers to analyze QoS constraints on message lifetimes as well as intermittent "ON-OFF" periods. We consider two varieties of applications – those that rely on *reliable* networking protocols (such as TCP) and those that rely on *unreliable* networking protocols (such as UDP). The nuances of sending messages over reliable/unreliable protocols are analyzed using *Queuing Network Models* [17]. We model the interactions in pub/sub protocols using continuous, ON-OFF, finite capacity and message expiration queueing centers. The detailed modeling provides a grounded foundation for QoS analysis of pub/sub interactions in the form of *Performance Modeling Patterns* (*PerfMP*), that may be composed with both reliable and unreliable messaging instances.

Experiments are conducted using the simulator we developed to study end-to-end response times and delivery success rates with reliable and unreliable communication patterns. We observe that even though there is a significant improvement in response time using the unreliable pattern, the delivery success rates cap at 64% despite increasing messaging lifetimes. This, however, is not in the case of reliable patterns, with increasing lifetimes resulting in guaranteed message reception with increased response times as the trade-off. Such analyses are crucial for application developers of complex systems such as TIM to accurately set interacting middleware parameters.

The core contributions of this paper are:

(1) A joint analysis of pub/sub protocols with QoS modeling to ensure timely end-to-end message delivery. (§2)
(2) Queuing Network Modeling of *publishers*, *subscribers* and *brokers* using M/M/1, ON-OFF (with intermittent available servers), Finite Buffer and Limited Lifetime Queueing Centers. (§3,§4)
(3) Analysis of reliable and unreliable middleware protocols in conjunction with the underlying networking stack to provide a unified framework for pub/sub modeling. This generates a set of performance modeling patterns (*PerfMP*) that may be reused across applications. (§5)
(4) End-to-end analysis through simulation based experiments for studying end-to-end response times and delivery success rates for both reliable and unreliable messaging patterns. (§6)

2 SYSTEM MODEL

The *Publish-Subscribe* interaction paradigm, is commonly used for content broadcasting/feeds. IoT middleware protocols such as MQTT [4] and AMQP [27], as well as tools and technologies such as RabbitMQ [22], Kafka [1] and JMS [26] follow the pub/sub paradigm. In pub/sub, multiple peers interact via an intermediate *broker*. Publishers produce *events* (or messages) characterized by a specific *filter* to the broker. Subscribers subscribe their interest for specific filters to the broker, who maintains an up-to-date list of subscriptions. The broker matches received events with subscriptions and delivers a copy of each event to each interested subscriber. There are different types of subscription schemes: *topic-based*, *content-based* and *type-based* [10]. For instance, in a topic-based pub/sub, events are characterized with a topic and subscribers subscribe to specific topics for receiving events.

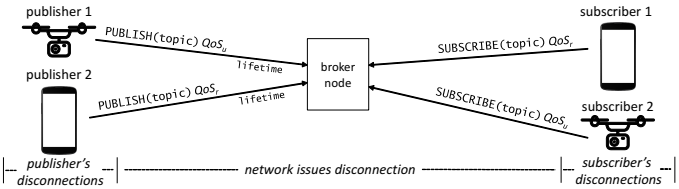


Figure 1: Pub/Sub QoS Features.

2.1 Pub/Sub QoS Features

A *Quality of Service (QoS) feature* is an agreement between a sender and a receiver defining the way that an event delivery is treated by the underlying communication infrastructure. We define the following QoS features that may be enabled in pub/sub messaging:

– QoS_{u}: is the feature for enabling *unreliable* interactions. For such interactions, **a sent event may or may not be received** by the receiver.
– QoS_{r}: is the feature for enabling *reliable* interactions. For such interactions, this feature guarantees that **the delivery of an event is verified** using ACKs, and retransmissions.

Depending on the requirements of an IoT application, a system designer is able to select between the above QoS features, which are provided by middleware pub/sub protocols (MQTT, AMQP, etc) for guaranteeing the corresponding response times or delivery success rates. We define the above QoS features in pub/sub systems for two different interactions: *i)* the *publisher - broker* interaction; and *ii)* the *broker - subscriber* interaction. As depicted in Fig. 1, a publisher is able to set a QoS feature (QoS_{u} or QoS_{r}) for publishing a particular event to the broker. Additionally, a subscriber subscribes using a specific QoS feature for receiving events by the broker. Accordingly, the QoS of an end-to-end interaction (from publisher to subscriber) can be downgraded if at least one of the two peers sets the lower QoS feature (e.g., the publisher publishes events with QoS_{r} and the subscriber subscribes to receive events with QoS_{u}). Additionally, for each event published the publisher may set `lifetime` period that represents the event's validity/availability in the pub/sub system (i.e., the broker or an application is responsible to delete an event when expires).

In what follows, we describe formally the proposed pub/sub system with QoS features and performance metrics.

2.2 Pub/Sub Formal Modeling

To mathematically represent the arrival rates of events, we use a topic-based subscription model, since it is efficient and simple in terms of event classification. Nevertheless, our approach can be used for any model where other classification techniques are applied. In content-based pub/sub systems, events are distinguished by the properties of the events instead of predefined criterion (i.e., topic name). Hence, our method must be modified based on the resulting probability distributions of the arrival rates after the filtering process in the broker node (possible via simulation based models).

Let V be the set of all topics in the system. We model the connectivity of pub/sub peers as follows: let ON and OFF be the states where the peer is connected and disconnected, correspondingly. A given peer, is connected (ON state) for an exponentially distributed time period with parameter θ_{ON} ($\theta_{\mathrm{ON}} = 1/T_{\mathrm{ON}}$). Upon the expiration of this time, the

peer disconnects (OFF state) and stops the transmission or reception of relevant events for an exponentially distributed time period with parameter θ_{OFF} ($\theta_{\text{OFF}} = 1/T_{\text{OFF}}$). Below we provide models representing our pub/sub system.

Publisher Model. Let P be the set of all publishers in the system. Each publisher ($p \in P$) forwards the published events to the broker when connected. p is defined by the tuple:

$$p = \left(id_{\text{p}}, V_{\text{p}}, \lambda_{\text{p}}, \texttt{lifetime}, qos_{\text{p}}, T^{\text{p}}_{\text{ON}}, T^{\text{p}}_{\text{OFF}}, K_{\text{p}}\right)$$

where id_{p} is the publisher's identifier, $V_{\text{p}} \subseteq V$ is the set of the topics on which p publishes events according to a Poisson process for each topic, λ_{p} is the input rate of the published events which is also Poisson (sum of Poisson processes for the set of topics V_{p}), `lifetime` is validity/availability time period of an event and is set upon its production, qos_{p} is the delivery method used by p for sending a particular event to the broker (i.e., QoS_{u} or QoS_{r}), T^{p}_{ON} and $T^{\text{p}}_{\text{OFF}}$ are the average periods where p connects and disconnects, and K_{p} is the maximum capacity for storing published events especially during the p's disconnection period (T^{p}_{OFF}).

Subscriber Model. Let S be the set of all subscribers in the system. Each subscriber ($s \in S$) receives relevant events from the broker when connected. s is defined by the tuple:

$$s = \left(id_{\text{s}}, V_{\text{s}}, \lambda_{\text{s}}, qos_{\text{s}}, T^{\text{s}}_{\text{ON}}, T^{\text{s}}_{\text{OFF}}, K_{\text{s}}\right)$$

where id_{s} is the subscriber's identifier, $V_{\text{s}} \subseteq V$ is the set of the topics that s has subscribed, λ_{s} is the input rate which is intermittent Poisson since s receive events when connected (events match s's topics V_{s}), qos_{s} is the delivery method used by s for receiving events by the broker (i.e., QoS_{u} or QoS_{r}), T^{s}_{ON} and $T^{\text{s}}_{\text{OFF}}$ are s's connection/disconnection periods, and K_{s} is the maximum capacity for storing incoming events.

Broker Model. The broker b receives the published events to several topics from publishers and forwards them to the corresponding subscribers according to their connectivity status and QoS feature. b is defined by the tuple:

$$b = \left(\lambda_{\text{b}}, N_{\text{s}}, K_{\text{b}}, T^{\text{b}}_{\text{ON}}, T^{\text{b}}_{\text{OFF}}\right)$$

where λ_{b} is the broker's input rate according to a Poisson process (dependent on the outputs of connected publishers), $N_{\text{s}} \subseteq S$ is the set of subscribers subscribed to the broker b and K_{b} is the maximum capacity for storing the published events to be delivered to the set of the subscribers subscribed (N_{s}). IoT middleware protocols are often lightweight, enabling the possibility of deploying software components on resource-constrained and mobile devices. Hence, broker components can be also deployed on such devices and thus, it is essential to consider their intermittent connectivity. T^{b}_{ON} and $T^{\text{b}}_{\text{OFF}}$ are the average periods where b connects and disconnects for receiving published events and forwarding events to subscribers.

In this paper, we analyze two specific QoS metrics – *Response Time* and *Delivery Success Rate*.

2.3 Pub/Sub QoS Analysis

End-to-end Response Time. To evaluate the end-to-end response time from p to s, denoted by $\Delta^{\text{p}}_{\text{s}}$, it is essential to calculate the delay of the published events at every peer and the broker they pass through. Such delays can be calculated by taking into account: *i)* the applied QoS features to the events sent by the publisher and received by the subscriber (i.e., QoS_{u} or QoS_{r}); and *ii)* every peer's (p and s), broker's (b) connectivity (i.e., T_{ON} and T_{OFF}).

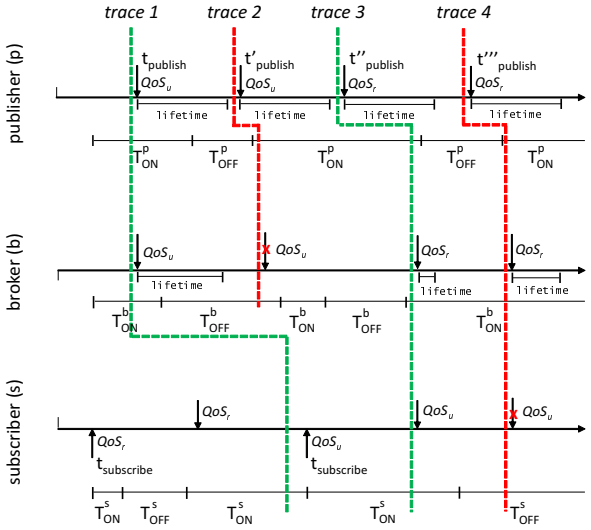


Figure 2: Analysis of Pub/Sub Interactions.

As depicted in Fig. 2, a publisher p publishes an event at t_{publish} timestamp. Independently of the publisher's connections/disconnections ($T^{\text{p}}_{\text{ON}}/T^{\text{p}}_{\text{OFF}}$), events may occur at any timestamp. Upon the occurrence of an event, in case the publisher is disconnected ($T^{\text{p}}_{\text{OFF}}$), events will be buffered for a maximum `lifetime` period until the publisher's next connection (T^{p}_{ON}) that will be sent to the broker (see Fig. 2). On the other hand, a subscriber s subscribes using a specific QoS feature (i.e., QoS_u or QoS_r) for receiving events at t_{subsribe} timestamp. In case the selected feature is QoS_r, during the subscriber's disconnections ($T^{\text{s}}_{\text{OFF}}$), events will be buffered at the broker for a maximum `lifetime` period – to be sent to the subscriber upon its next connection (T^{s}_{ON}). From p to s, delays occur due to event buffering in the following cases:

1. At the publisher's side, when events are produced at high rates (λ_{p}) and disconnections occur ($T^{\text{p}}_{\text{OFF}}$) at the same time.
2. At the publisher's side, when the QoS_{r} feature is selected for publishing events and b is disconnected ($T^{\text{b}}_{\text{OFF}}$).
3. At the broker's side, when the QoS_{r} feature is selected by s for receiving events and s is disconnected ($T^{\text{s}}_{\text{OFF}}$).
4. At the subscriber's (s) side, when receiving the incoming events in high rates (λ_{s}).

In addition to the above cases, to calculate the end-to-end response time ($\Delta^{\text{p}}_{\text{s}}$), it is essential to consider the expiration of events due to the applied `lifetime` period at each event.

Delivery Success Rate. The delivery success rate from p to s is denoted by $\Xi^{\text{p}}_{\text{s}}$. To identify *successful* or *failed* end-to-end interactions (from p to s), it is essential to analyze every possible combination of connections/disconnections, selected QoS features and lifetime periods, which are involved in such interactions. As depicted in Fig. 2, we distinguish 4 traces, which are analyzed in the following:

- *trace 1*: the event is published using the QoS_u feature; at the other side s is subscribed to receive events using the QoS_r feature. Upon the event's publication, p and b are at the ON state (T^{p}_{ON} and T^{b}_{ON}) and thus, the p-b interaction

is *successful*. Then, the event is kept at b for a maximum `lifetime` period and upon s's next connection (T^{s}_{ON} and s employs QoS_r) the p-s interaction is *successful*.

- *trace 2*: the event is published using QoS_u; at the other side s is subscribed to receive events using QoS_r. Upon the event's publication, p is OFF and thus the event is buffered for a maximum `lifetime` period. Upon p's next connection the event is transmitted independently of b's ON/OFF state (due to the employment of QoS_u by p). b is OFF and thus the p-s interaction is classified *failed*.

- *trace 3*: the event is published using QoS_r; at the other side s is subscribed to receive events using QoS_u. Upon the event's publication, p is ON but b is OFF and since the QoS_r feature is employed for publishing, the event is buffered for a maximum `lifetime` period. Upon b's next connection the event is transmitted to the broker and the p-b is *successful*. Afterwards, the event must be transmitted immediately to s (due to its QoS_u) and because of s's ON state, the p-s interaction is *successful*.

- *trace 4*: the event is published using QoS_r; at the other side s is subscribed to receive events using QoS_u. Similarly to *trace 2*, the event is buffered for a maximum `lifetime` period. Upon p's next connection (and if b is ON) the event is transmitted to b and the p-b interaction is *successful*. However, the event must be transmitted immediately to s (due to s's QoS_u) and because of s's OFF state, the p-s interaction *fails*.

In addition to the above traces, we must consider failed interactions due to `lifetime` periods. For instance, in *trace 1*, if the events are kept to the broker for a period higher to its lifetime and s is in the OFF state, then the b-s interaction will fail. Analyzing such traces in a formal way is not a trivial task. Additionally, we must take into account queueing delays for transmitting events through the underlying network infrastructure, as well as delays due to the high rate of event arrivals. Finally, we must distinguish between peers' disconnections occurred due to wireless network disconnections, or voluntary ones for energy saving purposes.

Pub/sub middleware protocols have been designed by taking into account the above limitations in order to facilitate developers building IoT applications. In this work, we particularly deal with the effect of mobility and network disconnections and the consequent effects on QoS. Our outputs are tuned taking into account `lifetime` and ON/OFF states. Note that aspects such as event duplication, ordered delivery or timely delivery [9] are not dealt with explicitly in this work. Likewise, end-to-end performance using routing or flows [5] [24] are also not evaluated. Further analysis of reliable/unreliable protocols is enunciated in the following.

3 RELIABILITY OF COMMUNICATION INFRASTRUCTURE

Middleware IoT protocols and APIs – such as MQTT [4], AMQP [27], Kafka [1] and JMS [26], follow the pub/sub paradigm and they mainly support asynchronous interactions. For instance, let us consider the interaction of two mobile peers; we demonstrate in the following the effect of disconnections on QoS. In this interaction, a *mobile publisher* produces events through multiple applications (apps). Each app disconnects from the network from time to time (e.g., for energy saving purposes), and the produced events are buffered until the next connection, upon which they are forwarded to the middleware (mdw) layer. On the other side, a *mobile subscriber* is able to receive events from multiple publishers; the events are distributed to multiple apps, whenever each app's connectivity (app-layer connection) allows it.

The mdw layer is responsible for handling and transmitting incoming events via the underlying network. Inside the network, additional disconnections may occur (defined as *mdw disconnections*) due to several reasons: *i)* broken session of the underlying protocol; *ii)* router crash/reboot; *iii)* wireless devices moving out of range; etc. Accordingly, event transmission may fail at the app/mdw layer. To enhance reliability, middleware protocols either rely on underlying protocol mechanisms, or they introduce additional ones, or in the case of pub/sub they incorporate broker nodes to decouple mobile peers. Thus, existing protocols can be categorized into:

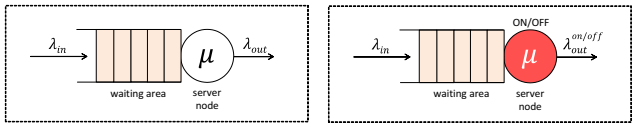
Unreliable middleware protocols: where guarantees for the delivery of events are missing. They typically build on top of the UDP unreliable transport protocol (e.g., CoAP non-confirmable). In UDP, two middleware nodes do not set up an end-to-end connection (publisher → broker, or broker → subscriber), as it is the case in TCP. Additionally, there is no confirmation for event delivery, and hence no event re-transmission in case of unsuccessful delivery. Thus, a **sent event may not be received**.

Reliable middleware protocols: where the delivery of each event is verified. They usually build atop the TCP reliable transport protocol [29]. Over TCP, two middleware nodes set-up or shut-down a reliable end-to-end connection (publisher → broker, or broker → subscriber) via 3-way or 4-way handshake, respectively. Based on TCP's mechanisms, **the delivery of each event is verified** using ACKs, timeouts and retransmissions. After the initial 3-way handshake, a session between the peers starts. During the session, intermediate routers can crash and reboot, wireless disconnections can occur, servers may shut down, and thus the session may break. There are several ways to detect such dropped connections in order to re-establish a TCP session. For instance, when sending out events without receiving ACKs; after several seconds the sender (e.g., publisher) considers the receiver (e.g., broker) is down and terminates the connection. In a different approach, some reliable protocols build atop UDP (e.g., CoAP/MQTT-SN) and add their own reliability mechanisms through additional acknowledgments (ACKs) or negative-acknowledgments (NACKs). Whether on top of TCP or on top of UDP, different levels of QoS may be provided (e.g., with MQTT, which adds its own reliability mechanisms on top of TCP). In our work, we correlate the intermittent ON/OFF periods of mobile peers, with the logical sessions of such reliable protocols.

After taking into consideration the above assumptions, in next section we provide a comprehensive approach for evaluating *end-to-end response times* and *delivery success rates* by relying on queueing theory.

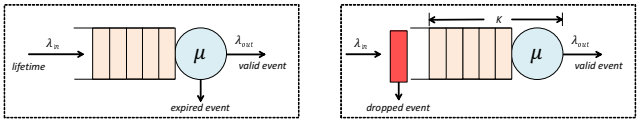
4 QUEUEING MODELS

To model end-to-end pub/sub interactions for performance analysis, we rely on queueing theory. In particular, we use simple inbound and outbound queues to estimate response



(a) Continuous queue. (b) Intermittent (ON/OFF) queue.

Figure 3: Continuous and intermittent queues.



(a) Applying `lifetime` periods. (b) Applying a buffer size (K).

Figure 4: Queues with event expirations and finite capacity.

times and event delivery success rates, as part of analytical or simulation models. Considering an end-to-end interaction between a sender and a receiver (e.g., publisher - broker), an *inbound queue* can be used to receive and process events (at the receiver's side) and an *outbound queue* to transmit them (at the sender's side). Each *queue* or *queueing center* serves events through a dedicated *server*. Each server supports a specific *service rate* (rate to process or transmit one event per time unit) denoted as μ. All queueing centers apply a first-come-first-served (FCFS) queueing policy. In this section, we define the individual queueing models used as part of the simulation queueing networks of Section 5.

Continuous Queueing Center. This queueing center models uninterrupted serving (transmission, reception or processing) of events as part of an end-to-end pub/sub interaction. It corresponds to the most common M/M/1 queue [17] (see Fig. 3a), featuring Poisson arrivals and exponential service times. An M/M/1 queueing center is defined by the tuple:

$$q_{\text{mm1}} = \left(\lambda_{\text{in}}, \lambda_{\text{out}}, \mu\right) \tag{1}$$

where λ_{in} is the input rate of events to the queueing center, λ_{out} is the output rate of events, and μ is the service rate for the processing of events. Let $\Delta_{\text{in}}^{\text{mm1}}$ be the mean response time. This is the time that an event remains in the system (corresponding to queueing time + service time). An analytical model for the mean $\Delta_{\text{in}}^{\text{mm1}}$ can be found in [17].

ON/OFF Queueing Center. To deal with the mobile peer's connections and disconnections we introduce the *Intermittent (ON/OFF)* queue, which is depicted in Fig. 3b. Events arrive according to a Poisson process with rate λ_{in}, and are placed in a queue waiting to be "served" (see Fig. 3b) with rate μ, which is exponentially distributed. We assume that the server is subject to an on-off procedure. That said, it remains in the ON-state for exponential time with parameter θ_{ON} ($\theta_{\text{ON}} = 1/T_{\text{ON}}$), during which it serves events (if any). Upon the expiration of this time the server enters the OFF-state during which it stops working (stops serving relevant events) for an exponentially distributed time period with rate θ_{OFF} ($\theta_{\text{OFF}} = 1/T_{\text{OFF}}$). Accordingly, an ON/OFF queueing center is defined by the tuple:

$$q_{\text{onoff}} = \left(\lambda_{\text{in}}, \lambda_{\text{out}}^{\text{onoff}}, \mu, T_{\text{ON}}, T_{\text{OFF}}\right) \tag{2}$$

where λ_{in} is the input rate of events to the queueing center, $\lambda_{\text{out}}^{\text{onoff}}$ is the output rate of events, and μ is the service rate for the processing of events (if any) during T_{ON}. The output process $\lambda_{\text{out}}^{\text{onoff}}$ is intermittent, because no event exits the queue during T_{OFF} intervals. Let $\Delta_{\text{in}}^{onoff}$ be the the mean response time (the time that an event remains in q_{onoff}). An analytical model for the mean $\Delta_{\text{in}}^{onoff}$ is provided in [7, 8].

Up to now, we have defined queueing models having buffers with infinite capacity and arriving events with infinite lifetime. This certainly affects the response time but also the rate of events successfully served over the total number of arriving events. However, modeling IoT protocols with such characteristics may not be realistic. For instance, upon a long disconnection period (e.g., 30 mins) of an IoT sensor, the produced data/events may exceed the sensor's buffer capacity and/or some of the oldest data may become obsolete for the receiving application/user. Accordingly, in this subsection we introduce the corresponding queueing model features that take into account the above constraints. These features can be applied on both continuous M/M/1 and ON/OFF queues.

Queueing Network with Event Expirations. A *queueing network* is a network of connected queues which can be used to model the performance of a system. For instance, in the context of our work, we model pub/sub interactions by creating queueing networks which consist of queueing models presented in the previous subsections. As depicted in Fig. 4a, events arrive with a rate λ_{in} in order to be processed in the first queue of the queueing network. An arriving event carries a `lifetime` period attributed to it upon its creation, which represents the event validity inside the queueing network. Hence, an event may enter the queueing network and as soon as its `lifetime` elapses, the event abandons the network and is considered as expired.

To consider an event as expired, we take into account the time the event spends in both queue and server at each queue. Assuming that a queueing network consists of a single M/M/1 queue, an event reneges if its service does not begin by a certain `lifetime` period (which includes its expected service demand as well). Based on the queueing theory literature, such a model is studied as an *M/M/1 queue with reneging* or *impatient customers* [20, 32]. In this work, we provide the simulation of the above M/M/1 model through our simulator. Furthermore, we enrich the ON/OFF queueing center to support *reneging* or *impatient customers*. Hence, system designers are able to create queueing networks that may include different queueing models (M/M/1 or ON/OFF) enriched with lifetime periods.

Queueing Center with Finite Capacity. This is a well known queueing model feature, where a specific buffer size is applied to the queue that ensures having max K events in the system (queue + server). This prevents from storing too many events for too long in devices with limited hardware capacity (memory, hard disk). In particular, as depicted in Fig. 4b, events arrive in the queue with λ_{in}. Before an event enters the queue the following condition is checked: $new\_queue\_size + event\_in\_service > K$. If the condition is true, the event is *dropped*. Otherwise, it enters the queue to be processed.

Based on the literature, an M/M/1 queue with finite capacity is notated as M/M/1/K [13]. In our models, we represent M/M/1 and the ON/OFF queues presented in the previous subsections with finite capacity by adding the system size (K) to the corresponding definition (see tuples 1 and 2).

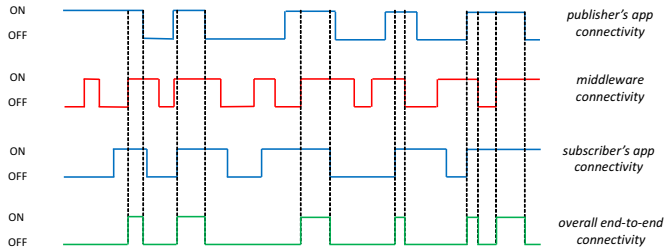


Figure 5: Overall end-to-end connectivity pattern.

Reliable and Unreliable Protocol Queueing Models. We use the above queueing centers as part of larger queueing networks in order to represent and evaluate the application and middleware layer of pub/sub systems. Estimated response times can be derived using both analytical and simulated models after composing larger queueing networks. Hence, the above queueing models can be applied to our simulation models and the trade-off between response times and delivery success rates can be evaluated.

To model the event transmission of **unreliable protocols**, we use an intermittent queue (ON/OFF, see Fig. 3b) at the app-layer of the mobile publisher representing its connectivity (e.g., voluntary disconnection). At the mdw layer we use continuous queueing centers (e.g., M/M/1, see Fig. 3a) with losses at the exit, for the processing of events regardless of the middleware/mobile broker's/subscriber's connection or disconnection. Either the event is successfully transmitted (in the former case) or it is lost (in the latter case).

To model the event transmission of **reliable protocols**, we use an intermittent queue at the app layer of the mobile publisher. To represent the end-to-end established session, we apply at the ON/OFF queue an overall connectivity pattern between the sender (e.g., publisher) and the receiver (e.g., broker). Determining such an overall pattern requires to take the intersection of the connectivity patterns of: *i)* the publisher's app; *ii)* the subscriber's app; and *iii)* the middleware. We illustrate this in Fig. 5. At the mdw layer we use continuous queueing centers to represent simply the processing/transmission times of events (end-to-end connectivity is represented by the above ON/OFF queueing center and there are no event losses).

5 PUB/SUB PerfMP

By following the assumptions presented in the previous sections, we now introduce performance models for both unreliable and reliable pub/sub interactions by using *Queueing Network Models*. We call these models *performance modeling patterns* (PerfMP), as they can be reused by system designers for evaluating the performance of IoT applications employing pub/sub protocols. In this section we provide the description of the pub/sub pattern.

Description. In pub/sub, a publisher publishes events to the broker and the subscriber receives them (through the broker). We assume that the subscriber is already subscribed to a specific `topic` in a broker and the publisher publishes events characterized by the same `topic`. For instance, a reporter posting news for a football team, which are received by another user. The *pub/sub pattern* is depicted in Fig. 6; it is used to model a reliable/unreliable pub/sub end-to-end interaction. Such a model evaluates the end-to-end response time and event delivery success rate of events, from the moment they are sent by the publisher's app, then they are received by the broker and are forwarded to the subscriber, until they are finally received by the subscriber.

At the publisher's side, for the reliable or the unreliable models, multiple apps produce events at the publisher's side (app layer). Each app may be disconnected (OFF-state – e.g., for energy saving purposes) and until its next connection (ON-state) the produced events are buffered in an ON/OFF queueing center. For any produced event a `lifetime` period is applied, which represents the event validity inside the queueing network. Let $\lambda_{\text{app}}^{p-in}$ be the input rate of events to the app's ON/OFF queueing center. The publisher's mdw layer accepts events from the specific app and from multiple other apps. Let $\lambda_{\text{apps}}^{p-in}$ be the input rate of events from other apps to the mdw layer.

Regarding the unreliable model, the publisher's mdw layer does not verify the successful transmission of events to the broker. Events are sent continuously without any knowledge of the disconnections of the mdw or the broker, which may result in losses. Hence, an event transmission is modeled using a continuous queueing center at the publisher's mdw layer, where the applied service rate μ_{tr} represents the transmission delay inside the network. Finally, additional event losses may occur due to event expirations at the app layer.

On the other hand, in the reliable model, the app's ON/OFF queueing center applies the overall end-to-end connectivity pattern (see Fig. 5). Thus, the app layer transmits events to the mdw as soon as an end-to-end connection (between the publisher and the broker) is established. Similarly to the unreliable model, an event transmission is modeled using a continuous queueing center at the publisher's mdw layer. In the reliable model the event reception is verified and, hence, event losses occur only in case of event expirations.

At the broker's side, for both the reliable/unreliable models, events arrive at the mdw layer through a continuous queueing center. These events may arrive from multiple publishers (see the additional flow λ_{oth}^{b}). Dropping of events occurs at the exit of the broker's input queue, depending on the subscriptions or due to event expirations (based on the `lifetime` period). In case an event is not dropped, it is forwarded to an output queue for its transmission to the corresponding subscriber. In the unreliable model, the transmission of events to the subscriber is done through an ON/OFF queueing center, which represents the broker's app-layer disconnections. This is the case where the broker is deployed in a mobile device and the transmission of events must be done based on the device's disconnections. Nevertheless, losses may occur due to middleware disconnections or event expirations. In the reliable model, the transmission is done through an ON/OFF queueing center, which represents the overall end-to-end connectivity between the broker and the subscriber (broker's, middleware, and subscriber's app connectivity), and losses may occur only due to event expirations. In this work, we focus on the end-to-end dissemination of events through a single broker by adapting QoS constraints found in the IoT. Interactions through multiple brokers and event duplications are considered in our previous work in [7].

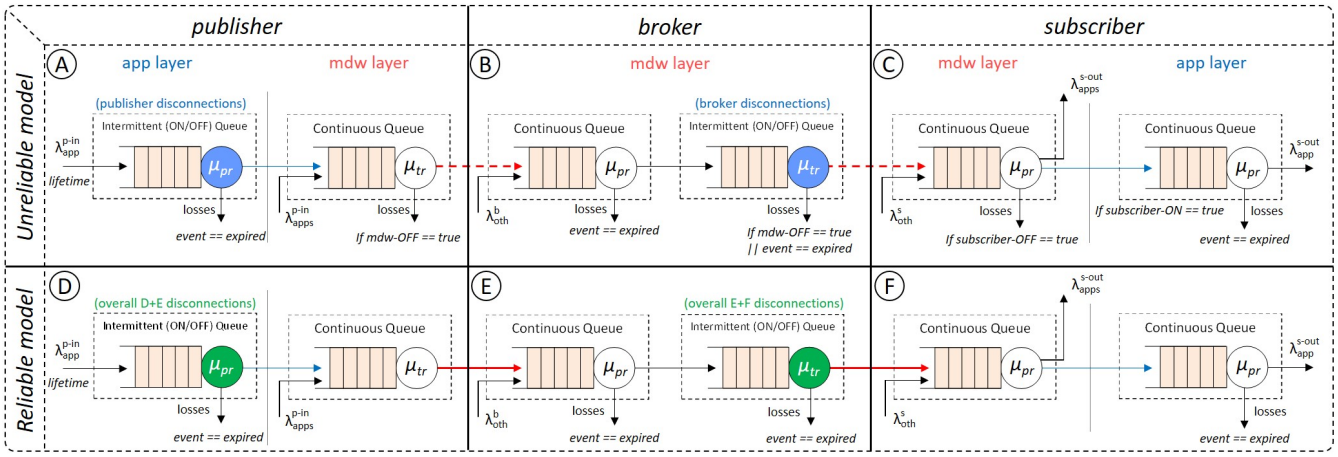


Figure 6: PerfMP for Pub/Sub Interactions.

Finally, at the subscriber's side, events arrive at the mdw layer through a continuous queueing center. These events may arrive from multiple other publishers (see the additional flow λ^{s}_{oth}) and be destined to multiple apps. Let $\lambda^{s-out}_{\text{app}}$ be the flow destined to the subscriber of interest. For the subscriber's app, a continuous queueing center is used to process events and detect possible event expirations.

The pub/sub pattern can be used to model several IoT protocols and messaging technologies. AMQP [27] and MQTT [4, 15] support pub/sub interactions. System designers are able to compose the corresponding queueing network that represents a pub/sub interaction (i.e., *publisher-broker* or *broker-subscriber* link) based on the applied QoS feature. Applying QoS_{u} in both interactions, results to the (A) → (B) → (C) queueing network (dotted arrows in Fig. 6). On the other hand, the (D) → (E) → (F) queueing network (solid arrows in Fig. 6) is used for QoS_{r} in both interactions.

Such features can be defined by relying on API of the corresponding protocol or messaging technology. For instance, based on the JMS API a subscriber can be defined as "non-durable" or "durable". For a non-durable subscriber, event losses occur upon its disconnections (mdw or app layer disconnections). Thus, the *broker-subscriber* link is unreliable (the QoS_{u} is applied upon subscription) and it can be evaluated using (B) → (C) queueing centers in Fig. 6. On the other hand, when a durable subscriber is disconnected, events are kept at the broker. Thus, the *broker-subscriber* link is reliable (the QoS_{r} is applied upon subscription) and it can be evaluated using (E) → (F) queueing centers in Fig. 6. It is worth noting that our pattern can model only event dissemination between publishers/subscribers. In our future work, we intend to identify the corresponding queueing network for the modeling of subscriptions as well.

6 EVALUATION RESULTS

Our simulator, MobileJINQS[1] implements the queueing models presented in Section 4. We then leverage MobileJINQS for creating simulation models that represent our proposed pub/sub performance patterns, such as the pub/sub *PerfMP* (see Section 5). Using such models, we perform statistical analysis and evaluate the trade-off between response times and delivery success rates.

In what follows, we use MobileJINQS to evaluate: *i)* the interaction between a reliable publisher that publishes events to a reliable subscriber; *ii)* the interaction between a reliable publisher that publishes events to an unreliable subscriber; and *iii)* the previous interaction (p reliable to s unreliable) with finite buffer capacities. We further show the applicability of our performance modeling for tuning the TIM system presented in the introduction.

6.1 Reliable p to Reliable s

In Section 5, we defined the pattern that models the performance of pub/sub interactions by incorporating the queueing models defined in Section 4. The resulting end-to-end queueing network, is depicted in Fig. 6. Based on the selected QoS features (see Section 2) by a system designer, an end-to-end queueing network can be created for evaluation. For our experimental setup we set the QoS_{r} for publishing events to the broker and the subscriber subscribes using the QoS_{r} feature for receiving events by the broker (see Fig. 1 of Section 2). Accordingly, the end-to-end queueing network is reliable (i.e., (D) → (E) → (F) in Fig. 6) where losses occur only due to event expirations – i.e., there are no losses due to disconnections. Based on Section 5, the resulting queueing network represents and evaluates *durable* subscribers, that can be implemented in pub/sub systems using the JMS API.

At the input of the pattern's queueing network, events arrive with rate $\lambda^{in}_{\text{app}} = 2$ events/sec (publishing rate). Each event is valid for a deterministic `lifetime` period and then discarded by the queueing network. We alternate between values of 10, 30 and 60 sec for the `lifetime` parameter. Arrival rates from/to multiple other apps going through the various queueing centers at the middleware layer are isolated; we assume that they have already been taken into account in the utilization of the servers of the queueing centers.

We parameterize the queueing network as follows: as already defined, the app layer's ON/OFF queueing center represents the overall end-to-end connectivity between the publisher and the broker. We set the total average publisher's connected + disconnected period to be $T^{\text{p}}_{\text{ON}} + T^{\text{p}}_{\text{OFF}} = 80$ sec. Experiments are performed by varying the T^{p}_{ON} and $T^{\text{p}}_{\text{OFF}}$

[1] https://github.com/boulouk/mobile-jinqs

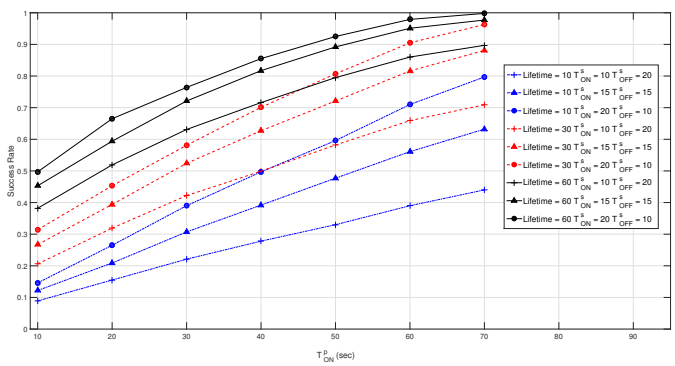


Figure 7: Success rates for reliable to reliable interactions with varying $T_{\text{ON}}/T_{\text{OFF}}$ and `lifetime`.

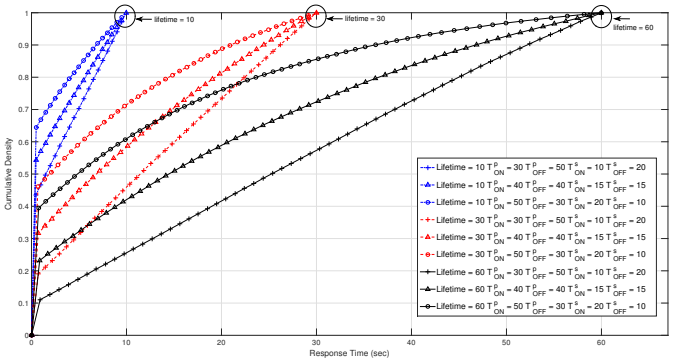


Figure 8: Response time distributions for reliable to reliable interactions with varying $T_{\text{ON}}/T_{\text{OFF}}$ and `lifetime`.

periods inside the 80 sec interval. To process the produced events and forward them to the mdw layer when connected, we apply a processing rate of $\mu_{\text{pr}} = 64$ events/sec. The applied processing rate is very low, since the app layer's queue is used locally only to forward events to the mdw layer. To transmit events to the broker, we apply a transmission rate of $\mu_{\text{tr}} = 32$ events/sec. The applied transmission rate can vary, depending on the bandwidth of the connection between the publisher and the broker. To process the incoming events at the broker's side, we apply (at the continuous queue) a processing rate of $\mu_{\text{pr}} = 64$ events/sec.

At the subscriber's side, we consider that peers remain always ON for receiving the subscribed events. However, middleware-layer disconnections may occur, which depend on the type of user mobility. Based on [3], such connection/disconnection (ON/OFF) periods vary in the scale of 0.5-1.5 min for *Traffic Information Systems*. Accordingly, we set the subscriber's total average connected + disconnected period to be $T^{\text{s}}_{\text{ON}} + T^{\text{s}}_{\text{OFF}} = 30$ sec. We apply this $T^{\text{s}}_{\text{ON}} + T^{\text{s}}_{\text{OFF}}$ period to the ON/OFF queue inside the broker transmitting events to the subscriber.

Experiments are performed by varying the T^{s}_{ON} and $T^{\text{s}}_{\text{OFF}}$ periods inside the 30 sec interval. During ON periods, events are transmitted to the subscriber with a transmission rate of $\mu_{\text{tr}} = 32$ events/sec (again this represents the network transmission delay based of the available bandwidth of the broker/subscriber link). Finally, at the subscriber's side we apply a processing rate of $\mu_{\text{pr}} = 64$ sec for the processing of incoming events by the continuous queueing center. To simplify our experimental setup, the broker node is assumed to be always in the state ON (e.g., deployed on Cloud).

Delivery Success Rates. In order to evaluate the effect of varying `lifetime` and connection/disconnection periods on delivery success rates (Ξ^{p}_{s}), we perform simulations after applying the above parameters to the queueing network of Fig. 6. At the publisher's ON/OFF queueing center, the T^{p}_{ON} period varies from 10 to 70 sec, increased by 10 sec at each experiment. Thus, $T^{\text{p}}_{\text{OFF}}$ equals the remaining time from the 80 sec total. At the subscriber's side, connections (T^{s}_{ON}) last 10, 15, 20 sec and disconnections ($T^{\text{s}}_{\text{OFF}}$) equal to the remaining, 20, 15, 10 sec. The rates of successful interactions are shown in Fig. 7 for various values of `lifetime`, and $T_{\text{ON}}/T_{\text{OFF}}$ periods for both the publisher and the subscriber. Using our simulator, we perform around 700,000 interactions for each experiment. As expected, increasing T_{ON} (of the publisher or of the subscriber) periods for individual `lifetime` values improves the success rate. On the other hand, the success rate is severely bounded by lifetime periods, especially for lower values. Hence, increasing lifetime periods from 10 sec to 30 sec is necessary to have a success rate of more than 73% for a connectivity of $T^{\text{p}}_{\text{ON}} = 50$ sec and $T^{\text{s}}_{\text{ON}} = 20$ sec.

Response Time vs. Success Rate. In order to study the trade-off between end-to-end response times (Δ^{p}_{s}) and delivery success rates (Ξ^{p}_{s}), we present cumulative response time distributions in Fig. 8. In comparison with the previous set of experiments, we keep the same intervals for the subscriber's connections/disconnections ($T^{\text{s}}_{\text{ON}}/T^{\text{s}}_{\text{OFF}}$), while the publisher's connections (T^{p}_{ON}) occur for 30, 40, 50 sec and disconnections ($T^{\text{p}}_{\text{OFF}}$) equal the remaining, 50, 40, 30 sec. Fig. 8, shows response times for successful interactions (i.e., we plot only interactions having response times lower than the lifetime period). From Fig. 8, lower lifetime periods produce markedly improved response time. For instance, with `lifetime` = 10 sec and equal $T_{\text{ON}}/T_{\text{OFF}}$ periods, 60% of the interactions complete within 1 sec. Comparing this to Fig. 7, with `lifetime` 10 sec, $T^{\text{p}}_{\text{ON}} = T^{\text{p}}_{\text{OFF}} = 40$ sec and $T^{\text{s}}_{\text{ON}} = T^{\text{s}}_{\text{OFF}} = 15$ sec, the probability of response time being less than 10 sec is 1 while the success rate is 0.32. By increasing the `lifetime` to 30 sec, the probability of response time to be less than 10 sec is 0.58 and the success rate is 0.65. Generally, these trade-offs confirm that higher lifetimes give better success rates but with higher response times.

6.2 Reliable p to Unreliable s

For this experimental setup we set the QoS_{r} for publishing events to the broker and the subscriber subscribes using the QoS_{u} feature for receiving events (see Fig. 1 of Section 2). Accordingly, the end-to-end queueing network connects the reliable part of the pub/sub pattern, to the unreliable one (i.e., (D) → (B) → (C) in Fig. 6) where losses occur due to both event expirations and middleware/subscriber disconnections. Based on Section 5, the resulting queueing network represents *non-durable* subscribers, that can be implemented in pub/sub systems using the JMS API. To parameterize this queueing network, we apply the same parameters as the ones applied in subsection 6.1. Since the subscriber uses the QoS_{u} feature, during its disconnections ($T^{\text{s}}_{\text{OFF}}$) events will

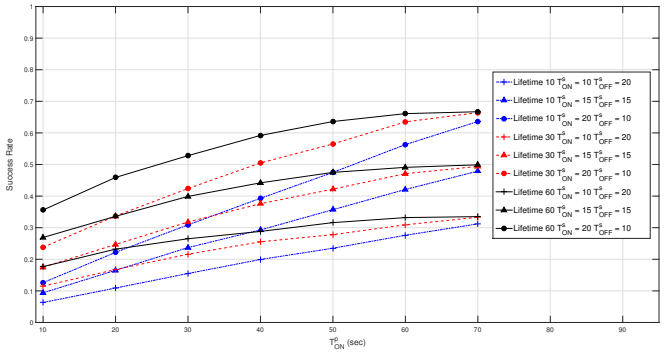


Figure 9: Success rates for reliable to unreliable interactions with varying $T_{\text{ON}}/T_{\text{OFF}}$ and `lifetime`.

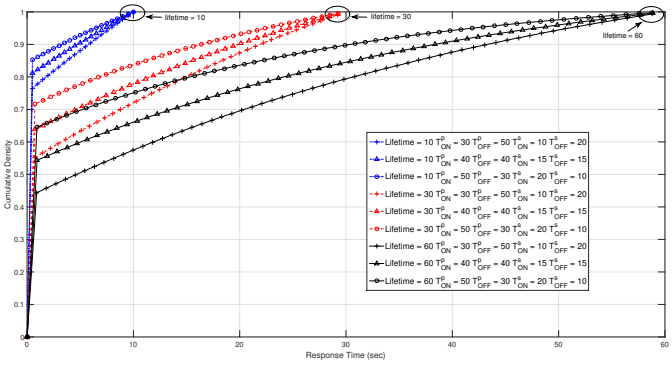


Figure 10: Response time distributions for reliable to unreliable interactions with varying $T_{\text{ON}}/T_{\text{OFF}}$ and `lifetime`.

be lost. As expected, the QoS of this end-to-end interaction is downgraded, which is illustrated in the following.

Delivery Success Rates. The rates of successful interactions are shown in Fig. 9 for various values of lifetime, and $T_{\text{ON}}/T_{\text{OFF}}$ periods for both the publisher and the subscriber. Similarly to the results of Fig. 7, increasing T_{ON} (of the publisher or of the subscriber) periods for individual `lifetime` values improves the success rate. However, the success rate is severely bounded not only by lifetime periods, but also due the subscriber's disconnections ($T^{\text{s}}_{\text{OFF}}$) – especially for lower values. For instance, increasing lifetime periods from 10 sec to 30 sec is necessary to have a success rate of more than 55% for a connectivity of $T^{\text{p}}_{\text{ON}} = 50$ sec and $T^{\text{s}}_{\text{ON}} = 20$ sec. It is worth noting that for higher values the success rate is severely bounded only by the subscriber's disconnections ($T^{\text{s}}_{\text{OFF}}$). For instance, for $T^{\text{p}}_{\text{ON}} = 70$ sec and $T^{\text{s}}_{\text{ON}} = 20$ sec, the success rate in not more than 64%, even if we vary the `lifetime` period to 10, 30 and 60 sec. This is also the case for $T^{\text{s}}_{\text{ON}} = 15$ sec and $T^{\text{s}}_{\text{ON}} = 10$ sec, where the success rate in not more than 50% and 33%, respectively ($T^{\text{p}}_{\text{ON}} = 70$ sec).

Response Time vs. Success Rate. Similar to subsection 6.1, we study the trade-off between end-to-end response times and delivery success rates by presenting cumulative response time distributions in Fig. 10. By using the same settings as the one used to plot the response time distributions of Fig. 8, Fig. 10 shows response times for successful interactions. These results show that lower lifetime periods in conjunction with higher $T^{\text{s}}_{\text{OFF}}$ (the subscriber's disconnection periods) produce markedly improved response time. This is due to

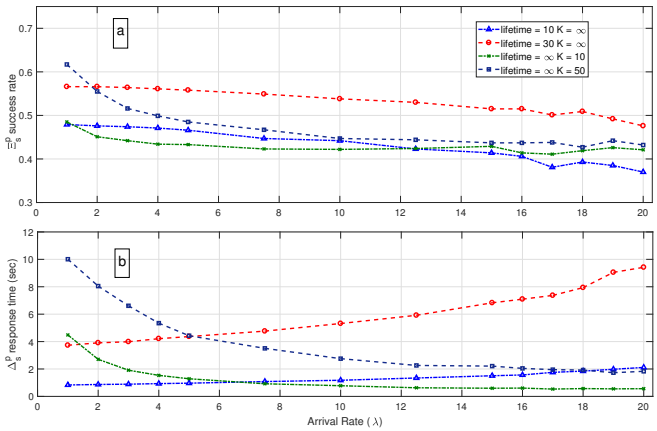


Figure 11: Success rates and mean response times for reliable to unreliable interactions with varying λ^{in}_{app}, K and `lifetime`.

the fact that during $T^{\text{s}}_{\text{OFF}}$ intervals, events are lost without causing queueing delays (due to the selected QoS_u feature at the subscriber's side). For instance, with `lifetime` = 10 sec and equal $T_{\text{ON}}/T_{\text{OFF}}$ periods, 80% of the interactions complete within 1 sec. On the other hand, by selecting the QoS_r feature at the subscriber's side, 60% of the interactions complete within 1 sec (see Fig. 8). Comparing the results of Fig. 10 to Fig. 9, with `lifetime` 10 sec, $T^{\text{p}}_{\text{ON}} = T^{\text{p}}_{\text{OFF}} = 40$ sec and $T^{\text{s}}_{\text{ON}} = T^{\text{s}}_{\text{OFF}} = 15$ sec, the probability of response time being less than 10 sec is 1 while the success rate is 0.3. By increasing the `lifetime` to 30 sec, the probability of response time to be less than 10 sec is 0.78 and the success rate is 0.38. Generally, these trade-offs confirm that higher lifetimes and lower $T^{\text{s}}_{\text{OFF}}$ periods give better success rates but with higher response times.

By comparing Fig. 9 (QoS_u subscriber) to Fig. 7 (QoS_r subscriber), it is important to note that for `lifetime` = 10 sec, success rates are approximately the same for lower T^{p}_{ON} values. For higher T^{p}_{ON} values there is a difference of maximum 10%. Comparing these results to Figs. 10 (QoS_u subscriber) and 8 (QoS_r subscriber) for `lifetime` 10 sec, the selection of the QoS_u feature at the subscriber side provide markedly improved response times.

6.3 Reliable p to Unreliable s with Finite Capacity

For this experimental setup we use the same end-to-end queueing network of the previous subsection (i.e., (D) → (B) → (C) in Fig. 6). In addition, we apply finite capacities (K) at each queueing center. Hence, losses occur due to event expirations, middleware/subscriber disconnections and finite capacities. To parameterize this queueing network, we apply the same processing and transmission rates (μ), as the ones applied in subsection 6.2. Then, we alternate between values of: *i)* 10 and 30 sec for the lifetime parameter; *ii)* 10, 50 for K at each queueing center. Regarding connections/disconnections, we apply the following values: $T^{\text{p}}_{\text{ON}} = 50$ sec, $T^{\text{p}}_{\text{OFF}} = 30$ sec, $T^{\text{s}}_{\text{ON}} = 20$ sec and $T^{\text{s}}_{\text{OFF}} = 10$ sec. Finally, for each lifetime and K parameters experiments are performed by varying the arrival rates (λ^{in}_{app}). We illustrate in the following the QoS trade-off when applying lifetime or K.

Delivery Success Rates. For each experiment we apply the following parameters: **1)** `lifetime` = 10 sec $K = \infty$; **2)** `lifetime` = 30 sec $K = \infty$; **3)** `lifetime` = ∞ $K = 10$; and **4)** `lifetime` = ∞ $K = 50$. The rates of successful interactions are shown in Fig. 11a. As expected, increasing `lifetime` and K values improves the success rate at any arrival rate. However, when increasing finite buffers for high λ^{in}_{app}, the success rate does not increase significantly. Generally, from Fig. 11a we notice that increasing: *i)* K at lower λ^{in}_{app} and *ii)* `lifetime` at higher λ^{in}_{app} – improves the success rate.

Response Time vs. Success Rate. In order to study the trade-off between success rates and response times, we present the mean values of response time in Fig. 11b for the same set of experiments. As expected, higher λ^{in}_{app} rates provide higher mean response times when applying `lifetime` periods. On the other hand when applying K values, low λ^{in}_{app} rates provide higher response times and high λ^{in}_{app} rates improves them. Generally, from Fig. 11b we notice that applying: *i)* K at higher λ^{in}_{app} and *ii)* `lifetime` at lower λ^{in}_{app} – improves the response time. Applying K at higher λ^{in}_{app} decreases the queueing delay in comparison to the application of `lifetime` values. By comparing Figs. 11a,11b, it is important to note that for high λ^{in}_{app} rates the response time is markedly improved when applying $K = 10$ and the corresponding success rate is slightly better in comparison to both response time and success rate when applying lifetime = 10 sec.

Note that though we have considered reliable p to unreliable s in these experiments, patterns provided in Fig. 6 may be similarly simulated for unreliable p to reliable s (i.e., (A) → (E) → (F) in Fig. 6). Through these experiments, we confirm that our analysis provides general guidelines for setting the lifetime, finite capacities and connection/disconnection periods to ensure successful interactions.

6.4 Discussion: TIM System Parameter Tuning

Linking back these results to the *Traffic Information Management* (TIM) system, application and middleware developers require tuning parameters such that freshness of delivered events are guaranteed. The TIM system is assumed to compose of both `smartphones`, built atop reliable middleware protocols as well as `vehicle-devices`/ `fixed-sensors`, built atop unreliable middleware protocols. This follows the *reliable* p to *unreliable* s model provided in Section 6.2.

Taking a publishing rate of $\lambda^{in}_{\text{app}} = 2$ events/sec, vehicles may connect and disconnect according to their profiles, yet expect timely information about crowd-sourced traffic congestion. The following profiles of users may be observed through historical statistics: publishing `smartphones` (T^{p}_{ON} = 50 sec, $T^{\text{p}}_{\text{OFF}}$ = 30 sec) and subscribing `vehicle-devices` (T^{s}_{ON} = 20 sec, $T^{\text{s}}_{\text{OFF}}$ = 10 sec). By relying on our analysis, an application designer may configure the `lifetime` periods of user access to 30 sec – using the data from Figs. 9 and 10, this guarantees that the `vehicle-device` will receive on average 58% of the posted notifications, within at most 16 sec of response time with a probability of 0.9. If these values are insufficient and the designer re-configures the `lifetime` to 10 sec, this now guarantees that the user will receive on average 48% of the posted notifications, within at most 3 sec of response time with a probability of 0.9.

As already pointed out, the above values can be configured for a publishing rate of $\lambda^{in}_{\text{app}} = 2$ events/sec. Using the data from Fig. 11, we notice that by increasing the $\lambda^{in}_{\text{app}}$ gradually up to 20 events/sec, the success rate of the subscriber decreases to 37% – for `lifetime` = 10 sec and 47% – for `lifetime` = 30 sec. On the other hand, the average response time of the subscriber increases to 2.1 sec – for `lifetime` = 10 sec and 9.49 sec – for `lifetime` = 30 sec. If these values are insufficient and the designer applies finite capacity buffers, i.e., $K = 10$ and $K = 50$, this guarantees that now the user will receive 42% of the posted notifications – for $K = 10$ and 44% – for $K = 50$. Furthermore, the average response time is now 0.43 sec – for $K = 10$ and 1.8 sec – for $K = 50$.

Such marked improvement in response time, with a marginal deterioration in success rates is non-trivial to analyze, without the use of toolkits such as *PerfMP* and *MobileJINQS*. This technique can be extended to other scenarios, where varying such parameters would provide QoS improvements.

7 RELATED WORK

In this section, we present our survey concerning the recent efforts in the design and evaluation of pub/sub systems. We begin with the work of Gaddah et al. [11, 12] where authors focus on the users' mobility inside pub/sub systems for investigating a pro-active caching approach. To design new hand-off management solutions, they consider a fixed network topology where transfer/caching of events/subscriptions between brokers occurs prior to subscribers' movement. To evaluate this approach, it is necessary to simulate the network topology and estimate several performance metrics (throughput, in this work), in order to compare them with other approaches. Authors represent the subscriber's mobility with connections and disconnections for randomly generated exponentially distributed times. However, publishing an event during subscriber's disconnection (OFF period) is considered as loss and is not waiting to the broker until the subscriber's reconnection. Finally, they utilize *continuous-time Markov chains (CTMC)* to express the subscriber's mobility and obtain the expected number of subscribers depending on the state (connected, disconnected, hand-off) for each broker. Performance metrics are derived through numerical methods whose solution demands high computational cost.

In [16, 23], a methodology for workload characterization and performance modeling of distributed pub/sub systems is presented. In this study, authors use *Queuing Petri Nets* for accurate performance prediction. While this technique is applicable to a wide range of systems, it relies on monitoring data obtained from the system and it is therefore only applicable if the system is available for testing. Furthermore, for systems of realistic size and complexity, QPNs would not be analytically tractable. Mühl et al. [21] present an approach for stochastic analysis of pub/sub systems employing identity-based hierarchical routing. This paper only considers routing table sizes and message rates as metrics.

Behnel et al. [5] provide an overview of QoS metrics and describe their applicability within the context of pub/sub systems. QoS metrics are related to latency, bandwidth and message priorities for subscriptions and single notifications,

but also for end-to-end flows. Authors evaluate a number of existing pub/sub systems (Hermes, IndiQoS, Choreca, etc) with regard to their QOS support. Furthermore, Corsaro et al. (2006) [9] study end-to-end QoS aspects for reliable message delivery, timely delivery and trust relationship.

Bellavista et al. [6] study state-of-the-art industrial and academic solutions focusing on their ability to support scalability and QoS requirements in smart city scenarios. Results show that different design and architectural details influence QoS in the presence of different event models, routing topologies, system scale and QoS parameters. By focusing on wide-scale topologies, Schroter et al. [24] provide analytical models for pub/sub systems employing different peer-to-peer and hierarchical routing schemes. The resulting analytical models address all the major performance metrics, such as notification delay, subscription delay and message rates. Authors present the exploitation of their models by performing performance prediction and capacity planning. In [7], we provide analytical models for wide-scale pub/sub systems. In particular, we analyze the end-to-end response time between mobile publishers and subscribers by focusing on the wide-scale analysis of intermediate brokers.

In this paper, we provide realistic Pub/Sub models by integrating QoS characteristics found in IoT applications. A generic set of patterns are provided when reliable and unreliable messaging protocols may be composed to derive end-to-end QoS outputs. System designers are able to evaluate end-to-end response times and delivery success rates and tune their IoT applications accordingly.

8 CONCLUSIONS

With the proliferation of mobile Internet of Things (IoT) devices, the Publish/Subscribe (pub/sub) interaction paradigm has been well employed due to the advantages of space–time decoupling. An important aspect of the pub/sub paradigm is the ability to set parameters such as message lifetimes, message buffer sizes and connectivity intervals, that significantly affects the end-to-end Quality of Service (QoS). In this paper, we provide a *Queuing Network Model* that captures the application and middleware layers of the pub/sub paradigm. By employing this model, we demonstrate how the end-to-end response times and delivery success rates may be studied, both for the case of reliable (e.g. TCP) and unreliable (e.g. UDP) underlying communication protocols. Our model is developed as a *performance modeling pattern (PerfMP)* for pub/sub, that may be tailored for multiple applications. Using an example of a *Traffic Information Management* system, this analysis if shown to help in accurately setting parameters to ensure timely delivery of fresh messages. Simulations performed using our simulator shows strong correlation between increasing lifetimes and delivery success rates for both reliable and unreliable pub/sub patterns.

ACKNOWLEDGMENTS

This work was supported by the Inria@SiliconValley International Lab and the research associate teams MINES/ACHOR.

REFERENCES

[1] Apache Kafka. 2018. http://kafka.apache.org/.
[2] L. Atzori, A. Iera, and G. Morabito. 2010. The Internet of Things: A survey. *Computer Networks* (2010).
[3] G. Bajaj, G. Bouloukakis, A. Pathak, P. Singh, N. Georgantas, and V. Issarny. Las Palmas, Gran Canaria, September 2015. Toward Enabling Convenient Urban Transit through Mobile Crowdsensing. In *IEEE ITSC*.
[4] Andrew Banks and Rahul Gupta. 2014. MQTT Version 3.1. 1. *OASIS standard* (2014).
[5] S. Behnel, L. Fiege, and G. Muhl. 2006. On quality-of-service and publish-subscribe. In *ICDCS Workshops*. IEEE.
[6] P. Bellavista, A. Corradi, and A. Reale. 2014. Quality of Service in Wide Scale Publish-Subscribe Systems. *IEEE Communications Surveys & Tutorials* (2014).
[7] G. Bouloukakis, N. Georgantas, A. Kattepur, and V. Issarny. L Aquila, Italy, April 2017. Timeliness Evaluation of Intermittent Mobile Connectivity over Pub/Sub Systems. In *ACM/SPEC ICPE*.
[8] G. Bouloukakis, I. Moscholios, N. Georgantas, and V. Issarny. Paris, France, May 2017. Performance Modeling of the Middleware Overlay Infrastructure of Mobile Things. In *IEEE ICC*.
[9] A. Corsaro et al. 2006. Quality of service in publish/subscribe middleware. *Global Data Management* (2006).
[10] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. 2003. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)* (2003).
[11] A. Gaddah. 2008. *A pro-active mobility management scheme for publish/subscribe middleware systems*. Ph.D. Dissertation. Citeseer.
[12] A. Gaddah and T. Kunz. 2010. Extending mobility to publish/-subscribe systems using a pro-active caching approach. *Journal of Mobile Information Systems* (2010).
[13] Donald Gross, John Shortle, James Thompson, and Carl Harris. 2008. *Fundamentals of queueing theory*. John Wiley & Sons.
[14] Pieter Hintjens. 2013. *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.".
[15] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate. 2015. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing* (2015).
[16] S. Kounev, K. Sachs, J. Bacon, and A. Buchmann. Orlando, FL, USA, 2008. A methodology for performance modeling of distributed event-based systems. In *IEEE ISORC*.
[17] E. Lazowska, J. Zahorjan, S. Graham, and K. Sevcik. 1984. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc.
[18] S. Lee, H. Kim, D. K. Hong, and H. Ju. 2013. Correlation analysis of MQTT loss and delay according to QoS level. In *ICOIN*. IEEE.
[19] P. Mohan et al. Raleigh, NC, USA, November 2008. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *ACM SenSys*.
[20] Aliakbar Montazer-Haghighi, J Medhi, and Sri Gopal Mohanty. 1986. On a multiserver Markovian queueing system with balking and reneging. *Computers & Operations Research* 13, 4 (1986), 421–425.
[21] G. Mühl, A. Schroter, H. Parzyjegla, S. Kounev, and J. Richling. 2009. Stochastic analysis of hierarchical publish/subscribe systems. In *Euro-Par*.
[22] Pivotal, "RabbitMQ". 2018. https://www.rabbitmq.com/.
[23] K. Sachs, S. Kounev, and A. Buchmann. 2013. Performance modeling and analysis of message-oriented event-driven systems. *Software & Systems Modeling* (2013).
[24] A. Schröter, G. Mühl, S. Kounev, H. Parzyjegla, and J. Richling. 2010. Stochastic performance analysis and capacity planning of publish/subscribe systems. In *DEBS*. ACM.
[25] Zach Shelby, Klaus Hartke, and Carsten Bormann. 2014. The constrained application protocol (CoAP). (2014).
[26] Sun Microsystems. JMS Specifications. 2018. http://www.oracle.com/technetwork/java/jms/index.html.
[27] OASIS Standard. [n. d.]. Oasis advanced message queuing protocol (amqp) version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf. ([n. d.]).
[28] Waze. 2018. https://www.waze.com/en.
[29] G. Wright and R. Stevens. 1995. *TcP/IP Illustrated*. Addison-Wesley Professional.
[30] XD. Traffic. 2018. http://inrix.com/xd-traffic.
[31] J. Yoon et al. PR, USA, June 2007. Surface street traffic estimation. In *ACM Mobisys*.
[32] Dequan Yue, Yan Zhang, and Wuyi Yue. 2006. Optimal performance analysis of an M/M/1/N queue system with balking, reneging and server vacation. *International Journal of Pure and Applied Mathematics* 28, 1 (2006), 101–115.