Performance Modeling of the Middleware Overlay Infrastructure of Mobile Things

Georgios Bouloukakis, Ioannis Moscholios, Nikolaos Georgantas, Valérie Issarny

Performance Modeling of the Middleware Overlay Infrastructure of Mobile Things

Georgios Bouloukakis*, Ioannis Moscholios†, Nikolaos Georgantas*, Valérie Issarny*
*MiMove Team, Inria Paris, France.
firstname.lastname@inria.fr
†Dept. of Informatics & Telecommunications, University of Peloponnese, Tripolis, Greece.
idm@uop.gr

***Abstract*—Internet of Things (IoT) applications consist of diverse Things (sensors and devices) in terms of hardware resources. Furthermore, such applications are characterized by the Things' mobility and multiple interaction types, such as synchronous, asynchronous, and streaming. Middleware IoT protocols consider the above limitations and support the development of effective applications by providing several Quality of Service features. These features aim to enable application developers to tune an application by switching different levels of response times and delivery success rates. However, the profusion of the developed IoT protocols and the intermittent connectivity of mobile Things, result to a non-trivial application tuning. In this paper, we model the performance of the middleware overlay infrastructure using Queueing Network Models. To represent the mobile Thing's connections/disconnections, we model and solve analytically an ON/OFF queueing center. We apply our approach to Streaming interactions with mobile peers. Finally, we validate our model using simulations. The deviations between the performance results foreseen by the analytical model and the ones provided by the simulator are shown to be less than 5%.**

***Keywords*—Middleware; Queueing Networks; Mobile Connectivity; Internet of Things**

I. Introduction

The mobile Internet of Things (IoT) comprises sensors and actuators that are heterogeneous with different operating (e.g., operating platforms) and hardware (e.g., sensor chip types) characteristics, hosted on diverse Things (e.g., mobile phones, vehicles, clothing, etc.). To support the deployment of such devices, major tech industry actors have introduced their own APIs and protocols to deal with the: *i)* limited hardware (energy, memory) resources, *ii)* low bandwidth, *iii)* noticeable underlying protocol overhead, etc. Existing (IP-based) IoT protocols, such as CoAP, MQTT, DPWS, XAMPP and ZeroMQ [1], [2], are being used today to face the above IoT limitations. Each protocol is placed between the application and transport layer, running on top of either the reliable TCP or the unreliable UDP, and implements its own mechanisms (e.g., additional retransmissions) to provide several guarantees to the application layer.

To guarantee specific response times and data delivery success rates between mobile Things, each protocol provides several Quality of Service (QoS) features. Initially, it inherits different characteristics from the underlying transport mechanisms. Subsequently, it supports different modes of message delivery. For instance, CoAP offers a choice between "Confirmable" and "Non-Confirmable" whereas MQTT support three choices ("Fire and forget", "Delivered at least once" and "Delivered exactly once") [3], [4]. Depending on the selected mode, response times and delivery success rates differ significantly. Furthermore, the devices that deploy such protocols can be mobile (e.g., smartphones with embedded sensors, wearable devices, etc), which results to higher response times due to the intermittent connectivity. The latter is due to resource saving purposes or it depends on the network coverage and capacity of a specific area. Finally, some IoT applications may need to guarantee the freshness of provided information by applying a (limited) lifetime period.

Under these constraints, an application designer should be able to analyze and configure certain system aspects (middleware QoS features, network and user connectivity, message lifetime period, allocated system resources) in order to guarantee the appropriate response time and delivery success rate between mobile Things. To investigate such features, it is essential to model the performance of middleware protocols by considering the above constraints.

In [2], [4], [5] the trade-off between response times and delivery success rates by using key IoT protocols is evaluated. However, such methods are protocol-specific and they limit the application designer upon the introduction of a new IoT protocol. Several existing efforts concerning the design and evaluation of mobile systems aim at guaranteeing QoS requirements under several constraints (e.g., intermittent availability, limited resources, etc). The evaluation methods used are derived mainly from the field of Queueing Theory. For instance, 2-Dimensional (2-D) Markov chains and quasi-birth-death (QBD) processes have been used in [6]–[9] to model the changing connectivity of mobile users when offloading computation or data to the cloud through either 3G or WiFi connections. However, actually applying these methods remains a complex and tedious task for application designers. On the other hand, regarding middleware protocols, Queueing Petri Nets (QPNs) have been used in [10] for accurate performance prediction. However, QPNs, while highly expressive in representing parallelism, are suited for small-to-moderate size systems and intake considerable computational resources [11].

In this paper, we model the performance of middleware protocols by relying on Queueing Network Models (QNMs) [12], [13]. QNMs have been extensively applied

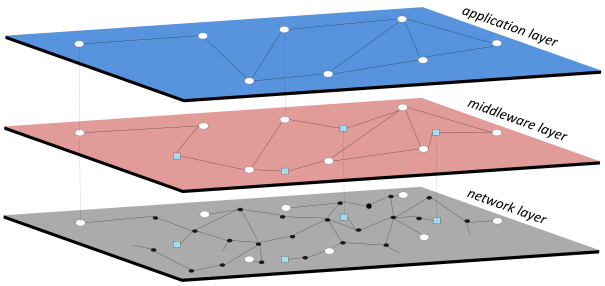

Fig. 1. Application/Middleware Overlays and physical network

to represent and analyze communication and computer systems and they have been proved to be simple and at the same time powerful tools for application designers with regard to system performance evaluation and prediction. Based on QNMs, middleware nodes (clients, servers, brokers, etc) are represented as queues, called *service centers* or *queueing centers*, and the exchanged messages as *jobs* served. In this work, we analyze a service center that represents an intermittently connected mobile Thing. This is modeled explicitly as an "ON/OFF queueing center" and is related to the Thing being either connected (ON) or disconnected (OFF). By analyzing this queueing center, we are able to derive useful performance metrics and evaluate the specific middleware platform that incorporates it. The key contributions of this work are:

- A methodology to model middleware protocols, which are used for the development of mobile IoT applications.
- An extensive analysis and performance metrics of the "ON/OFF queueing center", which can then be used as separate component inside queueing networks. Hence, we enrich the existing bibliography on QNMs and their solutions.
- A queueing network that models the performance of reliable Streaming middleware protocols.
- The validation of the proposed model via simulation.

The rest of this paper is organized as follows: Section II defines the overlay infrastructure introduced by Middleware protocols. In Section III, we provide the theoretical analysis of the "ON/OFF queueing center". A methodology to model the end-to-end message delivery of Streaming interactions is provided in Section IV. Finally, comparison with simulations is considered in Section V followed by conclusions in Section VI.

II. Middleware Overlay Infrastructure

In a typical IoT application scenario, mobile devices with embedded sensors (mobile peers) interact with each other, or they provide feedback to a centralized control center. Middleware protocols are built on top of common transport-level protocols (e.g., UDP, TCP) and they have introduced an additional layer. In this work, we focus on IP-based middleware protocols. Thus, we follow the TCP/IP model, where middleware protocols are placed between the application and transport layers.

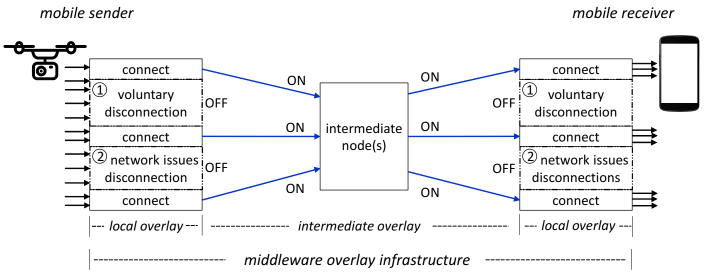

Fig. 2. Mobile peers' connectivity behaviour

Each middleware protocol implements its own mechanisms (e.g., additional retransmissions) in order to provide several guarantees to the application layer. Moreover, it is possible to introduce some additional intermediate nodes (e.g., a broker node in pub/sub protocols) in order to decouple the interactions between mobile Things. Depending on the role of each node (acting as server/producer/broker), they communicate directly through TCP or UDP sockets and they constitute the overlay network.

As depicted in Fig. 1, on the upper level, application (app) layer, Things interact with each other by relying on the provided features and mechanisms of the underlying overlay. On the middleware (mdw) layer, messages traverse the overlay topology based on their content and the specific middleware. On the lower (physical) lever, messages are routed through the physical network, which consists of physical routers and links that connect the middleware nodes in the overlay topology. Depending on the layer, we define the mobile peers' disconnections as follows: *i)* application layer disconnections occur on a voluntary basis (e.g., to save energy) independently from each app; and *ii)* middleware disconnections occur either due to network issues (e.g., wireless disconnection) or due to the receiver's crash/unavailability.

III. Queueing Models

We model the overlay infrastructure of the middleware layer using simple input and output queues. An *input queue* is used to receive and process messages and an *output queue* to transmit them. Each *queue* or *queueing center* serves messages through a dedicated *server*. Each server supports a specific *service demand* (time needed to process or transmit one message) denoted as D. Essentially, each queue is considered to be an M/M/1 queue, featuring Poisson arrivals and exponential service times. An M/M/1 queue is a *continues queueing center*, since there are no server interruptions when messages are served. Moreover, an M/M/1 queueing model usually is not the right model for most computer systems, but studying it develops the analysis techniques for more flexible models.

Indeed, to tackle with mobile peer's connections and disconnections we introduce the *ON/OFF queue*. Accordingly, we model the app and mdw layer of a mobile sender by using *input* and *output ON/OFF queues* in their local overlay to transmit messages. If the sender disconnects (state *OFF* in Fig. 2), messages remain at the *ON/OFF queue* until its next connection (state *ON* in Fig. 2), in which are transmitted to: *i)* the intermediate middleware nodes (if any exist); or *ii)* the receiver. A receiver is modeled as an *input M/M/1 queue* that receives messages from the access middleware component(s). Finally, an intermediate component (e.g., a broker of a pub/sub system)

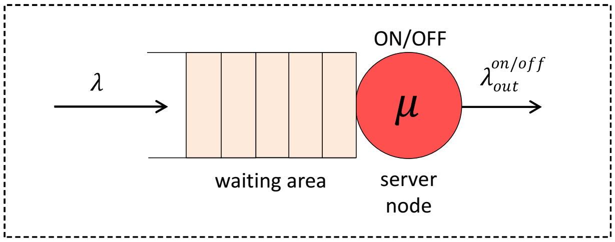

Fig. 3. An ON/OFF queueing node

can be modeled using multiple *input* and *output* (M/M/1 or ON/OFF) queues.

An M/M/1 queueing center ($q_{\mathrm{m/m/1}}$) is defined by the tuple:

$$q_{\mathrm{m/m/1}} = (\lambda, \lambda_{\mathrm{out}}, D)$$

where λ is the input rate of messages to the queueing center, λ_{out} is the output rate of messages, and D is the service demand for the processing of messages.

Based on standard solutions for M/M/1 queues [12], the time that a message remains in the system (queue+server) is given by:

$$T_{\mathrm{m/m/1}} \;=\; \frac{D}{1 \;-\; \lambda D} \tag{1}$$

A. ON/OFF Queueing Center

As already pointed out, the *ON/OFF queue* is utilized to model the local overlay of a mobile sender. In this section, we provide an extended description, the stated assumptions, a mathematical model, an extended analysis, and the performance metrics of the *ON/OFF queue.*

A.1. Description of the model

An *ON/OFF queue* represents a system having a single server, as depicted in Fig. 3. Messages arrive according to a Poisson process with rate $\lambda > 0$, and are placed in a queue waiting to be "served" (waiting area in Fig. 3). The *ON/OFF queue* processes messages through a dedicated *server* (server node in Fig. 3). The server supports a specific *service demand* (time needed to process one message) denoted as D, which is exponentially distributed with rate $\mu > 0$. The service station operates under the First-Come-First-Served (FCFS) queueing policy.

We assume that the server is subject to an on-off procedure. That said, it remains in the ON-state for an exponentially distributed time with parameter θ_{ON} ($\theta_{\mathrm{ON}} \;=\; 1/T_{\mathrm{ON}}$), during which it serves messages (if any). Upon the expiration of this time the server enters the OFF-state during which it stops working (stops serving relevant messages) for an exponentially distributed time with parameter θ_{OFF} ($\theta_{\mathrm{OFF}} \;=\; 1/T_{\mathrm{OFF}}$).

To model the performance of a component that sends messages under the effect of the above connectivity, we introduce the "ON/OFF queueing center".

A $q_{\mathrm{on/off}}$ queueing center is defined by the tuple:

$$q_{\mathrm{on/off}} = (\lambda, \lambda_{\mathrm{out}}^{\mathrm{on/off}}, D, \theta_{\mathrm{ON}}, \theta_{\mathrm{OFF}})$$

where λ is the input rate of messages to the queueing center, $\lambda_{\mathrm{out}}^{\mathrm{on/off}}$ is the output rate of messages, and D is the service demand for the transmission of messages (if any) during T_{ON}. The output process $\lambda_{\mathrm{out}}^{\mathrm{on/off}}$ is intermittent, because no message exits the queue during T_{OFF} intervals. Without loss of generality, we make the following assumption: if T_{ON} expires and there is a message currently being served, the server interrupts its processing and will continue in the next T_{ON} period.

Based on the description above, the probability of the server to be ON is given by:

$$P_{\mathrm{server}}(ON) = \frac{\theta_{\mathrm{OFF}}}{\theta_{\mathrm{ON}} + \;\theta_{\mathrm{OFF}}} \tag{2}$$

while the probability to be OFF is expressed as follows:

$$P_{\mathrm{server}}(OFF) = \frac{\theta_{\mathrm{ON}}}{\theta_{\mathrm{ON}} + \;\theta_{\mathrm{OFF}}} \tag{3}$$

A.2. The Analytical Model

The model can be described as a 2-D Markov chain. The state space diagram for this chain is depicted in Fig. 4. The states $(n, 1)$ and $(n, 0)$ refer to the case where there are n messages in the system (queue + server) and the server is online (ON) and offline (OFF), respectively. Clearly, if the server is always ON then we have an M/M/1 queue and the upper part of the 2D Markov chain (which describes state OFF) does not exist.

Based on this chain, the steady state probabilities $P_{n,k}$ ($n = 0, 1, ...$ while $k = 0, 1$) of the ON/OFF queue do not have a product-form solution since there are no backward transitions between the adjacent states $(n, 0)$ and $(n-1, 0)$. This is expected since messages cannot be serviced while the server is offline.

The global balance equations of the 2-D Markov chain have the form (rate into state (n, k) − rate out of state $(n, k) = 0$):

State(0,0): $\theta_{\mathrm{ON}} P_{0,1} - (\lambda + \theta_{\mathrm{OFF}})P_{0,0} = 0$
State(0,1): $\theta_{\mathrm{OFF}} P_{0,0} + \mu P_{1,1} - (\lambda + \theta_{\mathrm{ON}})P_{0,1} = 0$
State(1,0): $\lambda P_{0,0} + \theta_{\mathrm{ON}} P_{1,1} - (\lambda + \theta_{\mathrm{OFF}})P_{1,0} = 0$
State(1,1): $\lambda P_{0,1} + \mu P_{2,1} + \theta_{\mathrm{OFF}} P_{1,0} - (\lambda + \mu + \theta_{\mathrm{ON}})P_{1,1} = 0$
...
State(n,0): $\lambda P_{\mathrm{n-1,1}} + \theta_{\mathrm{ON}} P_{\mathrm{n,1}} - (\lambda + \theta_{\mathrm{OFF}})P_{\mathrm{n,0}} = 0$
State(n,1): $\lambda P_{\mathrm{n-1,1}} + \mu P_{\mathrm{n+1,1}} + \theta_{\mathrm{OFF}} P_{\mathrm{n,0}} - (\lambda + \mu + \theta_{\mathrm{ON}})P_{\mathrm{n,1}} = 0$
...

The above set of linear equations together with the normalization condition $\sum_{n=0}^{\infty} \sum_{k=0}^{1} P_{n,k} \;=\; 1$ provide a solution to all $P_{n,k}$. Having determined $P_{n,k}$ we can calculate the average number of messages in the system (server + queue), $E(n)_{on/off}$, via the formula:

$$E(n)_{on/off} = \sum_{n=1}^{\infty} n(P_{n,0} + P_{n,1}) \tag{4}$$

In the case of the M/M/1 queue the average number of messages in the system is given by [14]:

$$E(n) = \frac{\lambda}{\mu - \lambda} \overset{\rho=\frac{\lambda}{\mu}}{=} \frac{\rho}{1 - \rho} \tag{5}$$

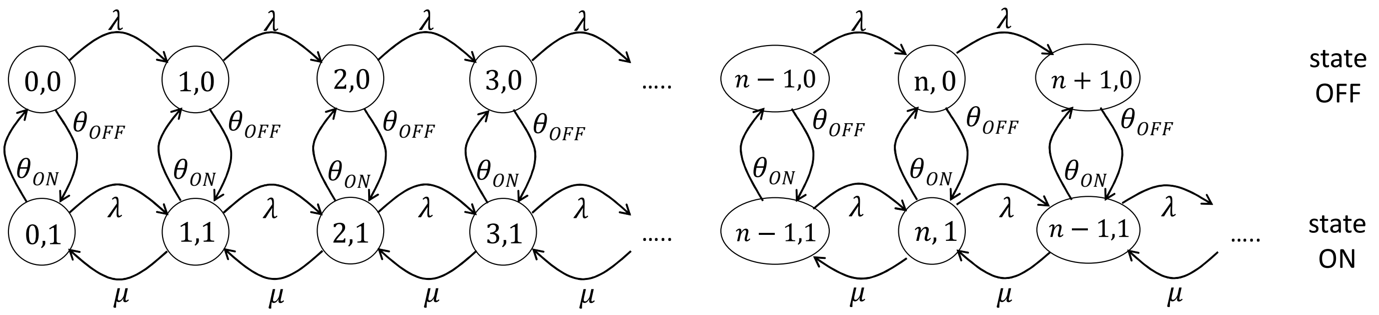

Fig. 4. 2-D Markov Chain

Since the actual service rate of the ON/OFF server is:

$$\mu^{'} = \mu P_{server}(ON) \tag{6}$$

We may assume that:

$$E(n)_{on/off} \approx \frac{\lambda}{\mu^{'} - \lambda} \overset{\rho^{'} = \frac{\lambda}{\mu^{'}}}{=} \frac{\rho^{'}}{1 - \rho^{'}} \tag{7}$$

Equation (7) underestimates (4) since in (7) we have not considered the population of messages that arrive in the system while the server is offline. Since the server returns to state OFF with probability $P_{\text{server}}(OFF)$ and remains offline for T_{OFF} time units, we can assume that the average number of messages that arrive in the system while the server is offline is given by $\lambda T_{\text{OFF}} P_{\text{server}}(OFF) = \rho_{\text{OFF}} P_{\text{server}}(OFF)$ and re-write (7) as follows:

$$E(n)_{on/off} \approx \frac{\rho^{'} + \rho_{\text{OFF}} P_{\text{server}}(OFF)}{1 - \rho^{'}} \tag{8}$$

Having determined $E(n)_{on/off}$ (either via (4) or in an approximate way via (8)), we can calculate the average system time (server + queue), $T_{\text{on/off}}$, via Little's law as follows:

$$T_{\text{on/off}} = \frac{E(n)_{\text{on/off}}}{\lambda} \tag{9}$$

In Section IV, we utilize the M/M/1 and ON/OFF queueing centers to model the performance of Streaming middleware protocols with mobile peers.

IV. Performance Model for Streaming Interactions

Sensors and actuators usually produce streams of data which require continuous processing. Middleware Streaming protocols, such as Websockets [15], Dioptase [16], XMPP [17], etc, support the transmission and processing of data streams through the peers involved in the interaction. Each peer can play either the *producer* role, which exposes data sources (e.g., sensor, database) as streams; or the *consumer* role, that acquires these streams. Additionally, the IoT is characterized by a network topology that may be unknown and highly dynamic, due to the mobility of Things or their short life span.

In this section, we model the performance of reliable streaming protocols, by taking into account the intermittent connectivity of mobile Things (producers and consumers). Towards this, we use the queueing models defined in Section III to form the queueing network of Fig. 5, which is used to model a reliable Streaming one-way interaction. Such an interaction is used to model the end-to-end delay of a message since is sent from an app of the producer, until the message is received by the consumer's app (we assume that the required stream session is already established between the consumer and the producer).

More specifically, multiple apps produce messages on the producer's side (app layer). Each app can choose to be disconnected for energy saving purposes. In case the app is disconnected, messages are buffered to an *ON/OFF queueing center*, until its next connection in which messages are forwarded to the mdw layer. Let $\lambda^{in}_{\text{app}}$ be the input rate of messages to the app's ON/OFF queueing center. The producer's mdw layer accepts messages from the specific app and from multiple other apps. Let $\lambda^{in}_{\text{apps}}$ be the input rate of messages from other apps to the mdw layer.

Reliable middleware protocols build on top of TCP [18]. This is the case when the producer and consumer set-up or shut-down a reliable end-to-end connection via 3-way or 4-way handshake, respectively. After the initial 3-way handshake, a session between the peers starts and a logical connection state is created. During the lifetime of a session, intermediate routers can crash and reboot, wireless disconnections can occur, servers may shut down, etc, and the session may become broken. There are several ways to detect such dropped connections in order to re-establish a TCP session. To model the message transmission of reliable Streaming protocols we use an *ON/OFF queueing center* on the producer's mdw layer, where the ON/OFF time periods follow the TCP connections/disconnections. Let D_{tr} be the service demand for each message to be transmitted (i.e., the delay introduced by the physical layer) to consumers.

Finally, on the consumer's side, messages arrive to the mdw layer through an *M/M/1 queueing center* and are distributed to multiple apps (e.g., an android app). Let λ^{out}_{c} be the output rate of messages to multiple apps. Except for the messages which arrive from the corresponding server, the mdw layer can accept messages from other servers. Let λ_{oth} be the rate of messages from other servers/producers.

Based on the queueing network of Fig. 5, (1) and (9), the end-to-end response time (T_{pc}) of a message sent from a producer's app to the consumer's app is given by:

$$T_{\text{pc}} = T^{\text{on/off}}_{\text{p−app}} + T^{\text{on/off}}_{\text{p−mdw}} + T^{\text{m/m/1}}_{\text{c−mdw}} \tag{10}$$

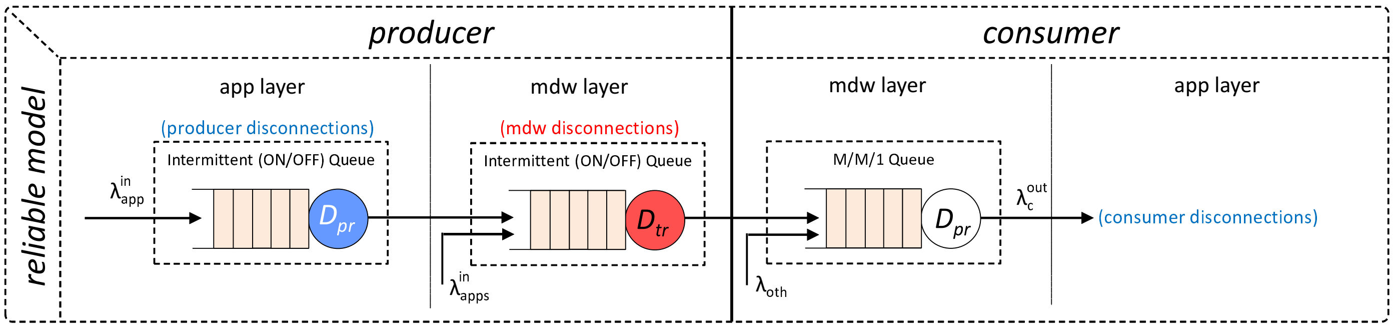

Fig. 5. Queueing Network for Streaming one-way reliable interactions.

where $T_{\mathrm{p-app}}^{\mathrm{on/off}}$ is the response time introduced due to the producer's app layer disconnections, $T_{\mathrm{p-mdw}}^{\mathrm{on/off}}$ is the response time introduced due to the middleware disconnections, and $T_{\mathrm{c-mdw}}^{\mathrm{m/m/1}}$ is the response time introduced due the to processing of the incoming messages at the consumer 's mdw layer.

V. Experimental Results

A. Simulation of the Queueing Models

We have developed a simulator that implements our queueing models. Our simulator, MobileJINQS[1], is an open-source library for building simulations encompassing constraints of mobile applications. MobileJINQS is an extension of JINQS, a Java simulation library for multiclass queueing networks [19]. JINQS provides a suite of primitives that allow developers to rapidly build simulations for a wide range of QNMs [12].

MobileJINQS retains the generic model specification power of JINQS, while it provides additional features of interest to mobile or other systems such as: (i) lifetime limitation for each customer entering a queue, (ii) intermittently available (on-off) queue server or server with variable service rate over time to represent mobile peers' behavior, and (iii) input flow with variable customer arrival rate over time to represent real input dataflow traces. Thus, an application designer is able to assign lifetimes, ON/OFF intervals, variable service rates and arrival rates following well-known probability distributions or real traces.

B. Analytical vs. Simulated Response Time

We utilize *MobileJINQS* to implement the *ON/OFF queueing center* described in subsection III-A. Mobile peers connect and disconnect in the scale of seconds/minutes to send/receive messages, depending on the application context. To represent such behavior, we set the ON/OFF system parameters as follows: *i)* the server remains in the *ON* and *OFF* states for exponentially distributed time periods $T_{\mathrm{ON}} = T_{\mathrm{OFF}} = 20/40/60$ sec, thus, the server changes its state every 20, 40 and 60 sec; *ii)* messages are processed with a mean service demand $D = 0.125$ sec; *iii)* there is sufficient buffer capacity so that no messages are dropped; and *iv)* messages arrive to the queue with a mean rate varying from 0.05 to 4 messages per sec ($\lambda_{\max} = 4$ messages/sec). By applying λ rates greater than 4 messages/sec, the system saturates. Using the above settings in our simulator, we run the system and derive the simulated curve of the mean response time for several λ rates as depicted in Fig. 6. The analytical results obtained by (9) and depicted also in Fig. 6, show the high accuracy of (9). For a service center where its server is always *ON*, the system does not saturate if $\lambda D < 1$. However, for the *ON/OFF queueing center* the system does not saturate if $1 - \rho^{'} < 1$ as indicated by the denominator of (8). Thus, this confirms that $\lambda_{\max} = 4$ messages/sec for this example. Indeed, by comparing the curves for the simulated and analytical response times, we notice small differences for rates equal to or higher than 3.5 messages/sec. This is acceptable, since the system is close to saturation at these rates.

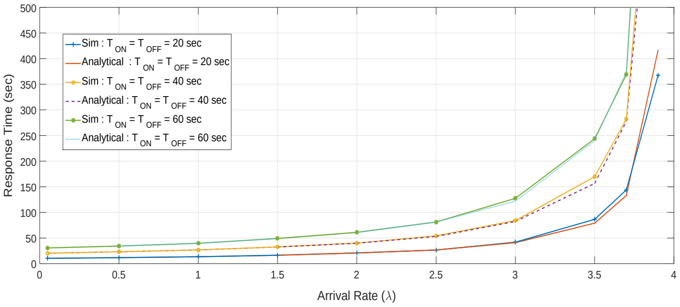

Fig. 6. Analytical vs. Simulated Response Times at the *ON/OFF queueing center*

C. End-to-end Response Time Evaluation

In Section IV, we model the performance of Streaming interactions by incorporating the intermittent connectivity of mobile peers. For our experimental setup, we use the end-to-end queueing network of Fig. 5 to evaluate the response time from a producer's app to a consumer. Thus, we utilize our proposed analytical solutions to study multiple application scenarios by varying the following parameters: λ, T_{ON}, T_{OFF} and D.

At the input of all the queueing centers, the messages arrive with rate $\lambda_{\mathrm{app}}^{in}$ (which is the rate of messages produced the at producer's app layer ON/OFF queue). Arrival rates from multiple other apps are isolated and not considered to the utilization of the servers for each queue. Due to lack of space, details about this isolation are not included in this paper.

To parameterize the queueing network, we use the parameters below. At the app's layer ON/OFF queueing

[1] xsb.inria.fr/d4d#mobilejinqs

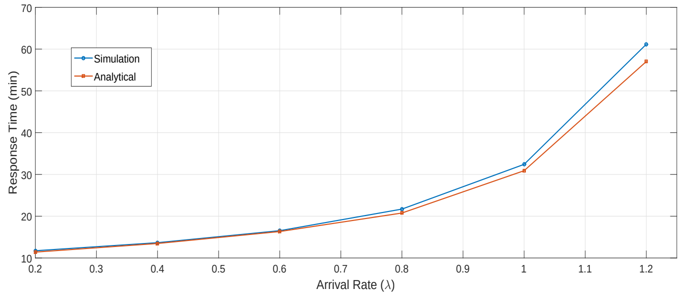

Fig. 7. Analytical vs. Simulated end-to-end response times for Streaming one-way interactions

center, the app is connected on average for $T_{\text{ON}} = 60$ sec to produce messages, and disconnected for $T_{\text{OFF}} = 10$ min to save energy resources to their device or to reduce the consumption of their monthly data plan. To process the produced messages and forward them to the mdw layer when connected, we apply a service demand of $D_{\text{pr}} = 0.0625$ sec. The applied service demand is very low, since the app's layer queue is used locally only to forward messages to the mdw layer.

The average middleware connection/disconnection periods depend on the type of user mobility. For example, these periods differ for pedestrians, vehicular, rail and metro passengers. In [20], we conclude that connectivity patterns of a path depend on both the network coverage and crowdedness of the metro the ON/OFF periods are in the scale of 1-3 min. Thus, at the producer's mdw layer ON/OFF queueing center, the mdw connects in average for $T_{\text{ON}} = 292$ sec and disconnected for $T_{\text{OFF}} = 78$ sec. To transmit messages to the corresponding consumer, we apply service demand $D_{\text{tr}} = 0.125$ sec. The applied service demand can be varied, depending the network type and the distance of the peers. Finally, at the consumer's mwd layer M/M/1 queueing center we apply service demand $D_{\text{pr}} = 0.0625$ sec for the processing of incoming messages.

Using our simulator, we create the queueing network as depicted in Fig. 5 and apply the above parameters. Furthermore, we apply the same parameters to (10). Subsequently, we run the queueing network for several arrival rates (0.2 - 1.2 msg/sec). Fig. 7 shows the resulting simulated and analytical response times in minutes. Confidence intervals of the simulation results are found to be very small (less than two order of magnitude) and are not presented in the figures. High differences are noticed when the system is close to its saturation ($\lambda_{\text{app}}^{in} = 1$ - 1.2 msg/sec). It is worth noting that the queueing network saturates because of the producer's app layer ON/OFF queue. Thus, using our approach, system designers are able to identify such bottlenecks.

VI. Conclusions

In this work, we model the overlay infrastructure of IoT middleware protocols using QNMs. QNMs are employed to provide an analytical solution for several performance metrics. To include the intermittent connectivity of mobile Things, we introduce the "ON/OFF queueing center". The *ON/OFF queueing center* is modeled as a separate queueing center, which we incorporate within a queueing network for Streaming interactions. We then validate the model via simulation. The analytical results obtained by the proposed model are highly satisfactory compared to simulation, proving the efficacy of our work. In future, we intend to extend this model by: *i)* applying time-to-live lifetime periods to each published message; *ii)* modeling unreliable protocols as well; and *iii)* introducing IoT protocols that follow other interaction styles, such as Client/Server, Publish/Subscribe and Tuplespace. Moreover, we aim to validate our model using real-word traces.

Acknowledgement

This work is partially supported by the associate team ACHOR and the H2020 project CHOReVOLUTION.

References

[1] V. Karagiannis *et al.*, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud Computing*, vol. 3, no. 1, pp. 11–17, 2015.

[2] K. Fysarakis *et al.*, "Which iot protocol? comparing standardized approaches over a common m2m application," Washington DC, USA, July 2016.

[3] S. Lee *et al.*, "Correlation analysis of mqtt loss and delay according to qos level," in *IEEE ICOIN*, Bangkok, Thailand, January 2013.

[4] N. De Caro *et al.*, "Comparison of two lightweight protocols for smartphone-based sensing," in *IEEE SCVT*, 2013.

[5] L. Durkop *et al.*, "Performance evaluation of m2m protocols over cellular networks in a lab environment," in *IEEE ICIN*, Paris, France, February 2015.

[6] F. Mehmeti and T. Spyropoulos, "Performance analysis of âĂIJon-the-spotâĂİ mobile data offloading," in *IEEE GLOBECOM*, Atlanta, USA, December 2013.

[7] K. Lee *et al.*, "Mobile data offloading: how much can wifi deliver?" in *Co-NEXT*, Philadelphia, USA, December 2010.

[8] T. Phung-Duc *et al.*, "A simple algorithm for the rate matrices of level-dependent qbd processes," in *ACM QTNA*, Beijing, China, July 2010.

[9] H. Wu and K. Wolter, "Tradeoff analysis for mobile cloud offloading based on an additive energy-performance metric," in *VALUETOOLS*. ICST, Bratislava, Slovakia, December 2014.

[10] S. Kounev *et al.*, "A methodology for performance modeling of distributed event-based systems," in *IEEE ISORC*, Orlando, USA, May 2008.

[11] M. Vernon *et al.*, *A comparison of performance Petri nets and queueing network models.* University of Wisconsin-Madison, Computer Sciences Department, 1986.

[12] E. Lazowska *et al.*, *Quantitative system performance: computer system analysis using queueing network models.* Prentice-Hall, Inc., 1984.

[13] F. Baskett *et al.*, "Open, closed, and mixed networks of queues with different classes of customers," *Journal of the ACM (JACM)*, 1975.

[14] D. Gross *et al.*, *Fundamentals of Queueing Theory.* John Wiley, 4th edition, 2008.

[15] I. Fette, "The websocket protocol," 2011.

[16] B. Billet and V. Issarny, "dioptase: data streaming middleware for the internet of things," *ERCIM News*, vol. 101, pp. 23–24, 2015.

[17] P. Saint-Andre, "Extensible messaging and presence protocol (xmpp): Core," 2011.

[18] G. Wright and W. Stevens, *TcP/IP Illustrated.* Addison-Wesley Professional, 1995.

[19] T. Field, "Jinqs: An extensible library for simulating multiclass queueing networks, v1. 0 user guide," 2006.

[20] G. Bajaj *et al.*, "Toward Enabling Convenient Urban Transit through Mobile Crowdsensing," in *IEEE ITSC*, Gran Canaria, Spain, September 2015.