Automating the evaluation of interoperability effectiveness in heterogeneous IoT systems

Georgios Bouloukakis, Nikolaos Georgantas, Ajay Kattepur, Houssam Hajj Hassan, Valérie Issarny

Automating the Evaluation of Interoperability Effectiveness in Heterogeneous IoT Systems

Georgios Bouloukakis[†], Nikolaos Georgantas[§], Ajay Kattepur[‡], Houssam Hajj Hassan[†], Valérie Issarny[§*]
{georgios.bouloukakis,houssam.hajj_hassan}@telecom-sudparis.eu,
nikolaos.georgantas@inria.fr, ajay.kattepur@ericsson.com
[†]Télécom SudParis, Institut Polytechnique de Paris, France
[§]Inria Paris, France
[‡]Ericsson AI Research, India

***Abstract*—Internet of Things (IoT) applications consist of diverse resource-constrained/rich devices with a considerable portion being mobile. Such devices demand lightweight, loosely coupled interactions in terms of time, space, and synchronization. IoT protocols at the middleware layer support several interaction types (e.g., asynchronous messaging, streaming, etc.) ensuring successful interactions between devices that use the same protocol. Additionally, they introduce different Quality of Service (QoS) delivery modes for data exchange with respect to available device and network resources. On the other hand, interconnecting heterogeneous IoT devices requires mapping both their functional and QoS properties. This calls for advanced interoperability solutions integrated with QoS modeling and analysis techniques. This paper introduces an automated synthesis of QoS-aware mediating artifacts. Such mediators enable the interconnection between IoT devices employing heterogeneous middleware protocols. Additionally, representative QoS models are synthesized. Leveraging these models, system designers can evaluate the effectiveness of the interconnection in terms of end-to-end QoS. We evaluate the usefulness of our approach through experimentation with a case study employing heterogeneous middleware protocols. In particular, we statistically analyze through simulations the effect of varying system parameters on the end-to-end QoS.**

***Index Terms*—IoT, QoS, Interoperability, Middleware**

I. Introduction

Internet of Things (IoT) devices penetrate homes, offices, community spaces as well as national infrastructures, and thus, by exploiting the generated data, there are now new opportunities to improve people's safety and quality of life via the creation of applications and services. Existing IoT protocols, such as CoAP, MQTT, XMPP and ZeroMQ [25], [23], are being used today to enable the exchange of IoT data. These protocols follow existing interaction paradigms, namely, Client/Server (CS), Publish/Subscribe (PS) and Data Streaming (DS), which implement various semantics in terms of time, space, and synchronization.

Besides the functional characteristics they offer, these protocols need to cope with the communication constraints of their diverse deployment environments, such as low network bandwidth or network disconnections. Each protocol inherits the Quality of Service (QoS) characteristics of the underlying transport mechanism (reliable TCP or unreliable UDP) and implements on top of it its own modes of message delivery. For instance, CoAP offers a choice between "confirmable" and "non-confirmable" message delivery, whereas MQTT supports three choices ("fire-and-forget", "delivered-at-least-once" and "delivered-exactly-once") [28], [18]. Depending on the selected mode, response times and message delivery success rates differ significantly: the employment of reliable delivery results in higher response times, while the use of unreliable delivery results in message losses.

Given their above characteristics, IoT protocols enable successful interactions when used in homogeneous settings. However, most often heterogeneous IoT devices need to be interconnected. In this case, both their functional and QoS properties need to be mapped. While several solutions have been proposed to deal with functional interoperability in IoT settings, the QoS dimension is usually ignored [19], [31], [17], [5]. In this paper, we advocate a holistic solution that, on one hand, enables automated synthesis of mediators that bridge heterogeneous protocols, and, on the other hand, generates QoS models for the composed end-to-end protocol. Based on these models, we propose to evaluate the interoperability effectiveness of the heterogeneous interconnection by analyzing the end-to-end QoS of the interconnection.

Regarding the functional dimension, we rely on the Data eXchange (DeX) interoperability framework [14]. DeX supports all of the CS, PS and DS interaction paradigms. The DeX Mediator Synthesizer (DeXMS) can generate a mediator for interconnecting two application components employing different IoT protocols that implement any two of the above interaction paradigms. The generated mediator is customized according to the data interfaces exposed by the two application components. For their interconnection, DeX abstracts these interfaces into a unifying DeX API.

In this paper, we enrich the DeX API with explicit QoS semantics (representing QoS semantics of CS, PS and DS protocols) and introduce what we call *QoS-aware DeX API*. Additionally, we develop our QoS modeling and analysis solution. First, we introduce a QoS algebraic framework that may be used to formally compose values of a QoS metric across heterogeneous interactions. This QoS algebra proposes operators that can be applied to different QoS properties,

*This paper is dedicated to the memory of Valérie, who was an esteemed researcher in the software engineering and distributed systems communities, and a dearest colleague. This work was initiated under her guidance.

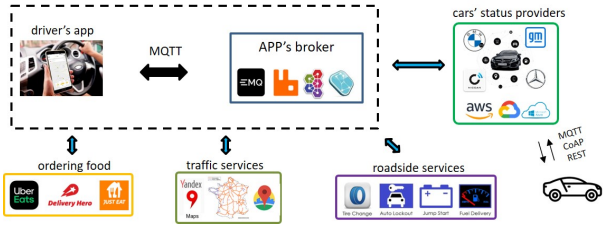


Fig. 1: IoT heterogeneity in the IoV scenario.

such as response time or reliability. Second, we model the performance of middleware protocols by relying on *Queueing Networks* (QNs) [26], [9]. QNs have been extensively applied to represent and analyze communication and computer systems and have proved to be simple and at the same time powerful tools for system designers with regard to system performance evaluation and prediction. We elaborate QNs to generically model protocols of the CS, PS and DS paradigms, as well as mediators that interconnect such protocols. We call these QNs *Performance Modeling Patterns (PerfMPs)*, as they can be applied to the modeling of various concrete protocols and their interconnections. While the QoS algebra provides a formal framework for the elaboration of the present (and potentially future) PerfMPs, it can be particularly useful in cases where PerfMPs cannot be parameterized. Such is the case of black-box components or networks, where a QoS property can only be externally measured.

Our holistic solution extends DeXMS so that it leverages: (i) the QoS-aware DeX API in the automated synthesis of mediators; and (ii) the QoS algebra as well as the PerfMPs in the automated synthesis of related QoS models. We call our solution *QoS-aware DeXMS*. By utilizing QoS-aware DeXMS, a system designer can create mediators to connect diverse IoT devices and assess interoperability effectiveness using generated end-to-end QoS models during the design phase.

The key contributions of this paper include:

- We introduce the QoS-aware DeX API, which represents functional and QoS semantics of CS, PS, DS (§III);
- We formalize QoS composition for IoT interactions via an algebraic framework (§IV).
- We elaborate performance modeling patterns for CS, PS and DS interactions (§IV).
- We enable automated synthesis of both mediators interconnecting heterogeneous IoT devices and related end-to-end QoS models (§V).

An overview of our approach is provided in (§II). We demonstrate how QoS-aware DeXMS can be leveraged for system tuning of heterogeneous interactions in an Internet of Vehicles (IoV) scenario in §VI. This is followed by a discussion of related work in §VII and conclusions in §VIII.

II. Overview

Motivating scenario: IoV. The highway systems in most countries are very sophisticated networks of road infrastructure and devices. As shown in Fig. 1, to improve the driving experience and public safety, road companies provide digital services for toll payment, real time traffic information, etc. In addition, roadside digital services related to parking capacity, charging/gas stations, are usually exposed as APIs, e.g., Google Places [2]. Numerous vehicles deploy a variety of sensors providing information related to vehicle characteristics (model, wheelbase) and status (speed, location, tire pressure) as well as trip and driver monitoring information (average fuel consumption, human/automated driver, alertness level, etc.). This has led to the emergence of the IoV [16] concept that enables data exchange between vehicles and roadside services using diverse technologies. For instance, charging an electrical vehicle requires 1-2 hours using fast-charging technology and up to 20 hours using Alternate-Current (AC) charging. Thus, proactive and deliberate planning is required especially in cases of long trips. In particular, given the distance of a trip, the vehicle's battery level, speed and location, an IoV application could anticipate and optimize the driver's stop plan (charging stations, restaurants, etc.).

To develop such IoV applications, it is essential to enable the exchange of data between heterogeneous vehicle sensors and highway/roadside digital services. In addition, due to the latency-based requirements of IoV applications (e.g., sources of data may become obsolete when estimating traffic) such data must be exchanged within limited latency bounds. Existing Cloud-based platforms have been introduced by vehicle manufacturers (BMW, Toyota, Tesla, etc.) to provide advanced car services such as parking assistant, concierge service, live traffic information, etc. While such systems improve the driving experience, they are vertical and siloed solutions, depending on the technology leveraged by the vehicle manufacturer. While these proposals aim to provide IoV services and applications, the plethora of platforms results in an additional layer of heterogeneity with diverse functional semantics (protocols/APIs & data formats), that are used from different vehicle vendors. In addition, IoV devices have diverse QoS semantics: (i) vehicles may experience *disconnections* due to the network coverage; (ii) IoV devices use different mechanisms for reliable message delivery; and (iii) information validity periods may be applied to ensure timely delivery.

High-Level Approach. Based on the functional and QoS semantics of an IoT application, a designer should be able to analyze and configure certain system aspects (message validity period, QoS delivery mode, etc.) to guarantee the appropriate system response time and delivery success rate. This paper presents an approach for *interoperability effectiveness evaluation* by: **(1)** Analysing the application's functional and QoS semantics; **(2)** synthesizing *QoS-aware mediators* that enable interoperability as well as the synthesis of *end-to-end performance models* and **(3)** performing statistical analysis via the simulation of end-to-end performance models for enabling system tuning. Accordingly, system designers are able to redefine QoS semantics of IoT applications to ensure timely delivery and successful interactions. By relying on our statistical analysis methodology, we demonstrate in §VI how system designers are able to tune our IoV application to ensure specific QoS requirements.

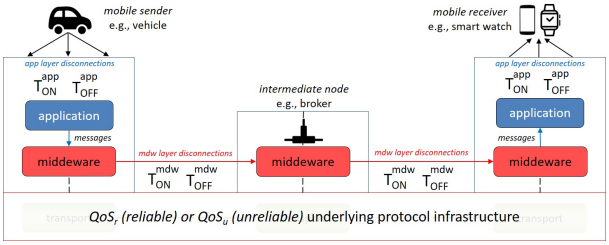


Fig. 2: Middleware interaction models.

III. Interconnecting Heterogeneous IoT Devices

IoT middleware protocols (MQTT, CoAP, XMPP, etc.) provide a number of features such as supporting different QoS delivery modes and they can be classified to different interaction paradigms, namely, CS, PS and DS. We leverage the DeX API [14], which is a set of primitives expressed as functions supported by the middleware. This API abstracts semantics of the basic interaction paradigms and therefore the semantics of the majority of existing IoT protocols. This paper presents a QoS-enhanced API that incorporates QoS-related parameters of interactions such as time validity of both messages (app-layer data) and control messages (e.g., subscriptions), the IoT devices intermittent connectivity and the IoT devices QoS delivery modes. Finally, we explain how the DeX API is leveraged to compose QoS-aware performance models for heterogeneous interactions in the IoT.

A. IoT Semantics for data exchange

By relying on [21], [6], semantics of interest include *space coupling*, *time coupling* and *synchronization coupling*. To express synchronization semantics of interactions, we define three *interaction types* and six role types:

one-way interaction: an IoT device takes the *sender*/*receiver* role; a sender sends a message without waiting for a response.

two-way asynchronous (async) interaction: an IoT device takes the *client*/*server* role; clients initiate a request to a server and then continue their processing; the server handles requests using callbacks and returns responses at some later point.

streaming interaction: an IoT device takes the *consumer*/*producer* role; the consumer requests to establish a dedicated session with the producer to send multiple messages that will be received by the consumer; both devices (or the consumer) can suspend/resume/terminate the session.

B. IoT Reliability for data exchange

IoT devices leverage middleware protocols to interact with each other using QoS delivery mechanisms, regardless of the possible disconnections. For instance, consider the interaction of two mobile peers where the *mobile sender* produces messages through multiple applications (see Fig. 2). Each application disconnects from the network from time to time (e.g., for energy saving purposes), and the produced messages are buffered until the next connection, upon which they are forwarded to the middleware (mdw) layer. At the other side, a *mobile receiver* is able to receive messages from multiple senders; the messages are distributed to multiple apps, whenever each app's connectivity (app-layer connection) allows it.

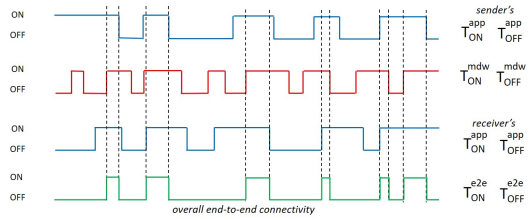


Fig. 3: Overall end-to-end connectivity pattern.

Let $\mathtt{T}_{\mathtt{ON}}^{\mathtt{app}}$ and $\mathtt{T}_{\mathtt{OFF}}^{\mathtt{app}}$ be the average app-based connection and disconnection time periods of mobile hosts, respectively.

The mdw layer is responsible to handle the incoming messages and transmits them via the underlying network. In the network, other disconnections may occur (*mdw disconnections*) because of: (i) broken session of the underlying protocol; (ii) router crash/reboot; (iii) wireless devices moving out of range; etc. Let $\mathtt{T}_{\mathtt{ON}}^{\mathtt{mdw}}$ and $\mathtt{T}_{\mathtt{OFF}}^{\mathtt{mdw}}$ be the average middleware connection and disconnection time periods between two hosts, respectively. Based on the above, message transmission may fail at the app or mdw layer. To enhance reliability, middleware protocols either rely on the underlying protocol mechanisms, or they introduce additional mechanisms, or they even incorporate middleware *broker nodes*.

Existing protocols can be categorized into: *i)* **unreliable**, where guarantees for the delivery of messages are missing and nodes do not set up an end-to-end connection (sender $\rightarrow$ broker-node, or broker-node $\rightarrow$ receiver, or sender $\rightarrow$ receiver). Let $\mathtt{QoS}_\mathtt{u}$ be the QoS delivery mode used from senders/receivers to unreliably send/receive messages; and *ii)* **reliable**, where the delivery of each message is verified and nodes set-up an end-to-end connection. Let $\mathtt{QoS}_\mathtt{r}$ be the QoS delivery mode used from senders/receivers to reliably send/receive messages (more details at [15]).

To represent the end-to-end established session, we apply an overall connectivity pattern between the sender and the receiver. Such a pattern requires to take the intersection of: *i)* the sender's $\mathtt{T}_{\mathtt{ON}}^{\mathtt{app}}$ and $\mathtt{T}_{\mathtt{OFF}}^{\mathtt{app}}$; *ii)* the receiver's $\mathtt{T}_{\mathtt{ON}}^{\mathtt{app}}$ and $\mathtt{T}_{\mathtt{OFF}}^{\mathtt{app}}$; and *iii)* the middleware's $\mathtt{T}_{\mathtt{ON}}^{\mathtt{mdw}}$ and $\mathtt{T}_{\mathtt{OFF}}^{\mathtt{mdw}}$. As shown in Fig. 3, we notate such an overall connectivity pattern as $\mathtt{T}^{\mathtt{e2e}}$.

C. QoS-aware Data eXchange (DeX) API

To enable data exchange between heterogeneous IoT devices, it is essential to introduce abstractions of their diverse interaction types, as well as the QoS-related parameters that IoT protocols offer. The basic interaction types can be implemented using the corresponding library implementation of every IoT middleware protocol. Based on [14], generic protocol primitives are introduced that model interaction types of peers following the CS, PS and DS interaction paradigms. As depicted in Fig. 4, in CS protocols, a client communicates directly with a server either by direct messaging (`send`, `receive`) or by a remote procedure call (RPC, two-way) through an operation. In PS protocols, publishers `publish` events characterized by a specific *filter* (e.g., topic) to the broker. Subscribers may choose to set up a callback function (`listen`) that will be triggered asynchronously by the broker when an event arrives (streaming). Finally, in DS protocols, a

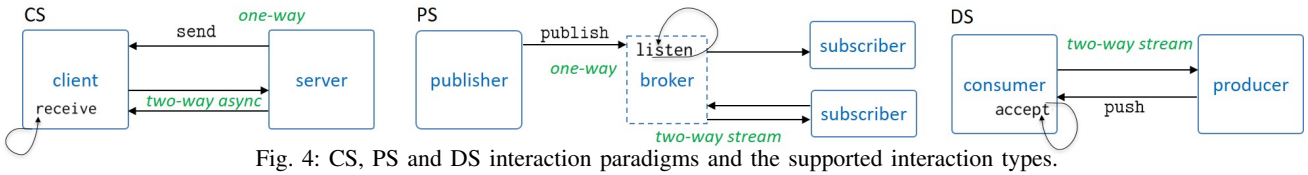


Fig. 4: CS, PS and DS interaction paradigms and the supported interaction types.

TABLE I: CS, PS and DS mapping to DeX primitives.

	DeX primitives	
CS	Server	Client
	`send(op, item, lf, QoS)`	`receive(op, *on_rec(), QoS, T^app); on_rec(item)`
PS	Publisher	Subscriber
	`publish(filter, event, lf, QoS)`	`listen(filter, *on_listen(), QoS, T^app); on_listen(event)`
DS	Producer	Consumer
	`push(str_id, data, lf, QoS)`	`accept(str_id, *on_accept(), QoS, T^app); on_accept(data)`
DeX	Sender	Receiver
	`post(scope, message, lf, QoS)`	`get(scope, *get_return(), QoS, T^app); get_return(message)`

consumer (typically) establishes a dedicated session with an *open stream* request sent to a producer. Upon the session's establishment, a continuous flow of data is pushed from the producer to the consumer (`push`, `accept`).

In this paper, as a first step, we enrich those primitives with the following QoS-related parameters. (i) `lifetime (lf)`: refers to emitted messages and characterizes both data availability/validity in time; (ii) T^{app}: average connected and disconnected periods at the client/consumer side (iii) QoS: refers to the QoS delivery mode (QoS_u or QoS_r) that is adopted from the employed middleware protocol at the server/producer or client/consumer side. Table I presents the enhanced primitives per interaction paradigm for one-way interactions. To enable QoS-aware interoperability between IoT devices employing protocols classified to CS, PS and DS, we enrich the DeX connector (presented in [14]) with the above QoS-parameters. In particular, as shown in Table I, the `post` and `get` high-level DeX primitives are mapped to CS, PS and DS primitives and QoS parameters for DeX one-way interaction.

In a DeX one-way interaction a *sender* sends messages using the `post` primitive with a validity period `lf` and a QoS delivery mode (QoS_u or QoS_r). At the receiver side, the `get` primitive is used to enable the reception of messages through the `get_return` primitive. In addition, the receiving entity defines the average app-layer connection/disconnection time periods (T^{app}), as well as the QoS delivery mode. Note that `post` and `get` operations are independent and have individual timestamps. We assume that application entities (undertaking the sender and receiver roles) enforce their semantics independently (no coordination). Note that in this paper we support DeX two-way asynchronous and streaming interactions using the DeX one-way primitives. In particular, the `post` and `get_return` primitives can model posting of requests, subscriptions and open streams, as well as posting items, events and data using the `message` generic parameter.

D. DeX Mediators

We now present how *DeX interactions* can be implemented by relying on the DeX API. In particular, the DeX API can be leveraged from developers to implement the `post` and

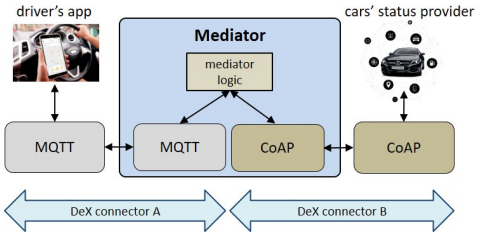


Fig. 5: Enabling *DeX interactions* via mediators.

`get` primitives using existing IoT protocols such as CoAP, MQTT, XMPP, etc. Then, the DeX connector model defines the composition of different primitives for implementing DeX interactions to the so called *DeX mediators* (DeXM). As depicted in Fig. 5, the mediator converts data coming from a vehicle device (in JSON format through MQTT) to be received from the diagnostics service (in XML format through CoAP). More details on DeXM can be found in [14].

While two heterogeneous IoT devices rely on mediators to interact with each other, the resulting end-to-end interaction may involve reliable/unreliable protocols, intermittent connected peers and data with specific availability. Therefore, the resulting performance might not satisfy the applications requirements. Existing interoperability approaches (§VII) focus on enabling the functional interoperability between heterogeneous IoT devices, while ignoring their end-to-end performance. In the next section, we leverage the QoS semantics of IoT devices to compose performance models. These can be further analyzed to ensure successful and timely interactions.

IV. QoS Modeling

This section introduces QoS domains and performance models for both unreliable/reliable CS, PS and DS protocols. We rely on Queueing Networks to provide *performance modeling patterns* (PerfMPs) that can be reused inside bigger compositions modeling end-to-end performance of IoT systems.

A. QoS Domain and Algebra

QoS metrics being probabilistic and multi-dimensional, accurate analysis of increments and composition rules are crucial. This section analyzes QoS domains that are of interest

TABLE II: Basic classes of QoS domains

QoS Metric	$\mathbf{D}$	$\leq$	$\oplus$	$\bigwedge$	$\bigvee$
δ: Response Time	$\mathbf{R}_+$	$<$	$+$	min	max
$\mathcal{R}$: Reliability	finite set $\mathbf{Q}$	$>$	$\bigvee$	max	min
$\mathcal{S}$: Success Rate	$\mathbf{R}_+ \in [0,1]$	$>$	$\times$	max	min

for interactions and the algebra for their composition. Specifically, in cases where detailed queuing network simulation models are unavailable (e.g. multi-vendor black-box setup), the observed metrics for various middleware paradigms may be composed using the algebra. Lower bound guarantees can also be expressed using the formal QoS representation.

QoS Domains. We review the basic domains of QoS for IoT:

δ: *Response Time* incorporates aspects of interaction latency for message delivery. For CS/DS interactions, timeliness concerns one-way message or two-way request-response latency. For PS, the latency between publication to a broker and subsequent coupled or decoupled delivery to peers is examined.

$\mathcal{R}$: *Reliability* incorporates the interaction reliability: a reliable interaction is either repeated until it is successful or reported as failed to the participating peers (typically the sender entities). Repeated transmissions have a negative impact on latency.

$\mathcal{S}$: *Success Rate* incorporates the interaction success rate that can vary for one-way/two-way interactions.

QoS Algebra. To aggregate metrics available from IoT interactions, we use an algebraic framework as introduced in [10]. This can handle random variables drawn from a distribution associated with a domain. A *QoS metric* q is a tuple:

$$q = (\mathbf{D}, \leq, \oplus, \bigwedge, \bigvee) \tag{1}$$

1) $(\mathbf{D}, \leq)$ is a QoS domain with a corresponding partially ordered set of QoS values.
2) $\oplus : \mathbf{D} \times \mathbf{D} \to \mathbf{D}$ defines how QoS increments by each new *action/operation*, like sending/receiving a message.
3) $(\bigwedge, \bigvee)$ represent the lower and upper lattice, meaning that any $q \subseteq \mathbf{D}$ has a unique greatest lower, least upper bound $(\bigwedge_q, \bigvee_q)$. When taking the *best* QoS with respect to the ordering $\leq$, we take the lowest QoS value, with $\bigwedge$. When synchronizing (e.g., with fork-joins), the operator $\bigvee$ takes the *worst* QoS as per the ordering $\leq$.

Basic classes of QoS domains are displayed in Table II and composed according to rules specified in Eq. 1. The use of this algebraic framework allows us to reason, in an abstract way, about the behavior of interaction paradigms and their effect on QoS performance. It specifies calculation of the QoS increments via the associated algebra with domains $\mathbf{D}$, partial order $\leq$, and operations $(\oplus, \bigvee, \bigwedge)$. When interconnecting heterogeneous IoT devices, the following operators are applied to each QoS domain to evaluate end-to-end QoS:

- Response Time: Here the $\oplus$ operator is used to compose end-to-end latency for interconnected components.
- Reliability: Composition of heterogeneous components with varying reliability is performed using the $\bigvee$ operator.
- Success Rate: Composition of heterogeneous components with varying success rates are performed using the $\times$ operator in the domain range $[0, 1]$.

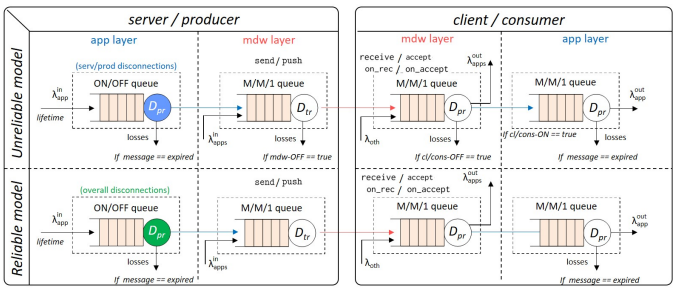


Fig. 6: PerfMP for CS/DS one-way interactions.

B. CS/DS Performance Modeling

CS/DS PerfMP. Based on CS interactions, a server sends a notification message to a client (e.g., push notification). In DS interactions, a producer sends a stream of data to a consumer. Regarding the latter, we assume that the required stream session is already established between the consumer and the producer. The *CS/DS one-way* pattern is depicted in Fig. 6; it is used to model a reliable/unreliable CS/DS one-way interaction. To model the message transmission of CS/DS reliable protocols, we use an intermittent queue (ON/OFF [13]) at the app-layer of the mobile sender. At the mdw-layer, we use M/M/1 queues to simply represent the processing/transmission times of messages (end-to-end connectivity is represented by the above ON/OFF queue and there are no message losses). On the other hand, CS/DS unreliable protocols apply only the app-layer connectivity (e.g., voluntary disconnection) in the ON/OFF queue. At the mdw layer, we use M/M/1 queues (with losses at the exit) for the processing of messages regardless of the middleware/mobile receiver's connection or disconnection. Either the message is successfully transmitted (in the former case) or it is lost (in the latter case).

The CS/DS PerfMP (see Fig. 6) evaluates the end-to-end response time and message delivery success rate for this interaction. For both the reliable/unreliable models, multiple apps produce messages at the server's/producer's side (app layer). Each app may be disconnected (e.g., for energy saving purposes) and until its next connection the produced messages are buffered in an ON/OFF queue. For any produced message a `lifetime` period is applied, which represents the message validity inside the queueing network. Let $\lambda^{in}_{\text{app}}$ be the input rate of messages to the app's ON/OFF queue. Let $\lambda^{in}_{\text{apps}}$ be the input rate of messages from other apps to the mdw layer. Regarding the unreliable model, the server's/producer's mdw layer does not verify the successful transmission of messages to the client/consumer. Messages are sent continuously without any knowledge of mdw or client/consumer disconnections, leading to potential losses. Hence, a message transmission is modeled using an M/M/1 queue at the server's/producer's mdw layer, where the applied service demand D_{tr} represents the delay inside the network. Finally, additional message losses may occur due to message expirations at the app layer.

On the other hand, in the reliable model, the app's ON/OFF queue applies the overall end-to-end connectivity pattern (see Fig. 3). Thus, the app layer transmits messages to the mdw as soon as an end-to-end connection (between the server/pro-

ducer and the client/consumer) is established. Similarly to the unreliable model, a message transmission is modeled using an M/M/1 queue at the server's/producer's mdw layer. Nevertheless, in the reliable model the message reception is verified and, hence, message losses occur only in case of message expirations. Finally, at the client's/consumer's side, messages arrive to the mdw layer through an M/M/1 queue. These messages may arrive from several servers/producers (see the additional flow λ_{oth}) and be destined to multiple apps. Finally, let $\lambda_{\text{app}}^{out}$ be the flow destined to the client/consumer of interest. For the client's/consumer's app, an M/M/1 queue is used to process messages and detect message expirations; its service demand (D_{pr}) represents the app's processing time.

The CS/DS one-way pattern can be used to model several IoT protocols. DPWS, OPC UA, REST, XMPP, Websockets support CS/DS one-way interactions [23], [25]. These protocols rely on TCP's delivery mechanisms and thus, they can be modeled using the reliable pattern where the overall end-to-end connectivity follows TCP's sessions. On the other hand, CoAP transmits messages over the unreliable UDP protocol. However it supports two built-in features: "non-confirmable" and "confirmable". The "non-confirmable" can be modeled using the unreliable pattern since it does not guarantee any message delivery. The "confirmable" feature supports message re-transmissions using ACKs and NACKs and thus, it can be modeled using the reliable pattern.

CS/DS QoS algebra. We use the algebra notations to model the algebraic composition of DS/CS interactions. For the unreliable model, the DS *response time* domain includes end-to-end delays introduced by the producer/ consumer (client/server in CS). Here we refer to the queues in Fig. 6 with `producer_app`, `producer_mdw`, `consumer_mdw` and `consumer_app`. The *response time* operator is defined as:

$$\delta_u^{ds} = \bigwedge(\texttt{lifetime}, (\delta_{\texttt{producer\_app}} \oplus \delta_{\texttt{producer\_mdw}} \oplus \delta_{\texttt{consumer\_mdw}} \oplus \delta_{\texttt{consumer\_app}})) \quad (2)$$

Here, the response time is the minimum of the `lifetime` and the summation of delays in message passing. The DS reliability depends on the reliability between `producer_mdw` and `consumer_mdw`. This is defined as follows:

$$\mathcal{R}_u^{ds} = \mathcal{R}_{\texttt{producer\_mdw}-\texttt{consumer\_mdw}} \quad (3)$$

Finally, the DS success rate depends on the intermediary success rates between application and middleware as follows:

$$\mathcal{S}_u^{ds} = (\mathcal{S}_{\texttt{producer\_app}} \times \mathcal{S}_{\texttt{producer\_mdw}} \mathcal{S}_{\texttt{consumer\_mdw}} \times \mathcal{S}_{\texttt{consumer\_app}}) \quad (4)$$

For the reliable model, we refer again to Fig. 6. The response time domain would include the end-to-end delays introduced by the producer and consumer as follows:

$$\delta_r^{ds} = \bigwedge(\texttt{lifetime}, (\delta_{\texttt{producer\_app}} \oplus \delta_{\texttt{producer\_mdw}} \oplus \delta_{\texttt{consumer\_mdw}} \oplus \delta_{\texttt{consumer\_app}})) \quad (5)$$

The reliability domain depends on the reliability of the intermediate transmission schemes. Here it is always reliable:

$$\mathcal{R}_r^{ds} = 1 \quad (6)$$

Finally, the success rate domain depends on the intermediary success rates. Here, only the applications can drop messages:

$$\mathcal{S}_r^{ds} = (\mathcal{S}_{\texttt{producer\_app}} \times \mathcal{S}_{\texttt{consumer\_app}}) \quad (7)$$

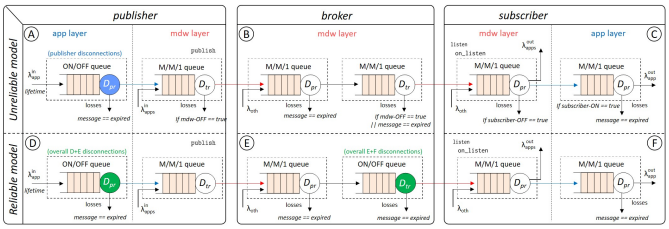


Fig. 7: PerfMP for PS one-way interactions.

C. PS Performance Modeling

PS PerfMP. A publisher publishes messages to the broker and the subscriber receives them (via the broker). We assume that the subscriber is already subscribed to a specific `filter` in a broker and the publisher publishes messages characterized by the same `filter`. The *PS one-way pattern* is depicted in Fig. 7; it is used to model reliable/unreliable Publish/Subscribe one-way interactions. Such a model evaluates the end-to-end response time and message delivery success rate of messages, since they are sent from the publisher's app, then they are received by the broker and are forwarded to the subscriber, and finally they are received by the subscriber.

At the publisher's side, the model is similar to the server's/producer's side in the *CS/DS PerfMP* (using PS primitives). At the broker's side, for both the reliable/unreliable models, messages arrive to the mdw-layer via an M/M/1 queue. These messages may arrive from multiple publishers (λ_{oth}). Dropping of messages occurs at the exit of the broker's input queue, depending on the subscriptions or due to message expirations (`lifetime`). If a message is not dropped, it is forwarded to an output queue for its transmission to the corresponding subscriber. In the unreliable model, the transmission of messages to the subscriber is done via an ON/OFF queue, 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 messages must be done based on the device's disconnections. Nevertheless, losses may occur due to mdw disconnections or message expirations. In the reliable model, the transmission is done via an ON/OFF queue, 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 message expirations. Finally, subscriber's model is similar to the client's/consumer's side in the *CS/DS PerfMP*.

The *PS one-way* pattern models several IoT protocols and messaging technologies, e.g., AMQP [33] and MQTT [8], [25]. AMQP and MQTT rely on TCP's delivery mechanisms and introduce additional built-in features for the end-to-end (from the publisher to the subscriber) message delivery such as *QoS 0, 1 & 2 levels*. We model AMQP, MQTT and their built-in QoS features using the reliable model. Using JMS [4], a "non-durable" subscriber may loose messages during its disconnection periods (mdw or app layer disconnections). Thus, the *broker-subscriber* link is unreliable and it can be evaluated using Ⓑ - Ⓒ queues in Fig. 7. On the other hand, when a "durable" subscriber is disconnected, messages are kept at the broker (reliable link) and thus, the *broker-*

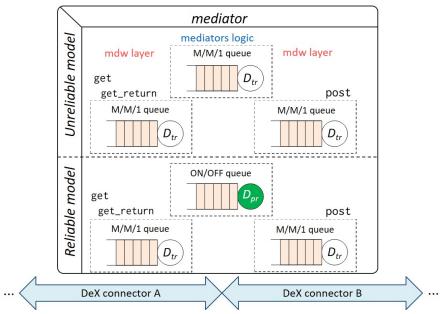


Fig. 8: PerfMP DeX Mediators.

subscriber link can be evaluated using Ⓔ - Ⓕ queues in Fig. 7.

PS QoS algebra. We provide a high level view of the QoS algebraic composition of PS interactions. Here we refer to the queues in Fig. 7 with `publisher_app`, `publisher_mdw`, `broker_mdw`, `subscriber_mdw` and `subscriber_app`. For the unreliable model, the response time domain would include the end-to-end delays introduced by the publisher, broker and subscriber. The response time is the minimum of the `lifetime` and the summation of delays in message passing. The reliability domain depends on the reliability of the intermediate links between `publisher_mdw`, `broker_mdw` and `subscriber_mdw`. If any of the links are unreliable, the end-to-end link is unreliable. The success rate domain depends on the intermediary success rates between application and middleware. For the reliable model, we refer again to Fig. 7 to produce the QoS metrics. The response time domain would include the end-to-end delays introduced by the producer and consumer. The reliability domain depends on the reliability of the intermediate transmission schemes – it is always reliable. The success rate domain depends on the intermediary success rates where only the applications can drop messages.

D. DeX Mediator Performance Modeling

Mediator PerfMP. To define the PerfMP of mediators, we rely on the mapping of DeX primitives to CS, PS and DS primitives defined in Table I. A mediator acts as a *proxy server* that receives messages from a heterogeneous IoT device using protocol A, converts these messages to protocol B, and finally sends them to the second heterogeneous device. This is performed by combining two DeX connectors. At the receiver part of DeX connector A, an M/M/1 queue is used to receive messages via the `get` primitive (mapped to `receive`/`listen`/`accept`). Similarly, at the sender part of DeX connector B, an M/M/1 queue is used to model the response time of messages sent via the networking infrastructure. Finally, DeX connectors merge their app-layers using a single queue which models the conversion logic of the mediator (M/M/1 for unreliable, ON/OFF for reliable).

Mediator QoS algebra. We now model the QoS algebraic composition of interactions passing via mediators. For the unreliable case, the response time domain would include the delays introduced by the `sender_mdw`, `mediator` and `receiver_mdw`. Here, the response time is the minimum of the `lifetime` and the summation of delays in message mediation. The reliability domain depends on the reliability of the intermediate links between `receiver_mdw`, `sender_mdw` and `mediator`. If any of the links are unreliable, the end-to-end link is unreliable. Finally, the success rate domain depends on the intermediary success rates between middleware connectors. For the reliable case, the reliability depends on the reliability of the intermediate transmission schemes – it is always reliable. Finally, the success rate domain depends on the intermediary success rates. As only the application can drop messages, the success rate is 1.

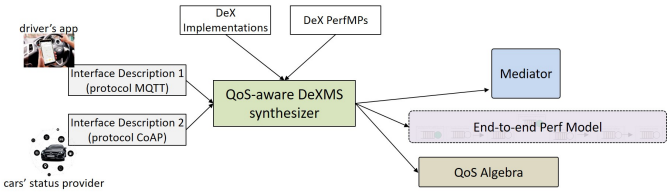


Fig. 9: QoS-aware DeXMS synthesizer.

V. End-to-end Composition

This section introduces, the ***Qos-aware DeXMS synthesizer*** which combines the QoS-aware DeX API, the PerfMPs and the QoS algebra for enabling the interoperability of heterogeneous IoT devices as well as evaluating their interoperability effectiveness. In our running example (described in §II), to enable functional and timely data exchange between the driver's application (MQTT protocol used) and the car's status provider (CoAP protocol), the synthesizer acts as following: *i)* identifies the supported interaction types, DeX APIs and defines the required DeX connectors; *ii)* maps the MQTT topics with the CoAP resources; and *iii)* identifies and maps their PerfMPs and QoS algebra based on their interaction paradigms and unreliable/reliable QoS delivery modes.

As the functional mapping process is described in §III-D, this section describes how the *end-to-end PerfMP* and *end-to-end QoS algebra* are composed. Based on protocol defined in Fig. 5, the synthesizer provides as output the composition of PerfMPs for PS, CS interaction paradigms, and the mediator's PerfMP. The resulting *end-to-end PerfMP* is shown in Fig.10. With regard to the *end-to-end QoS algebra* using the algebraic compositions of CS, DS, PS and the mediator's (defined in §IV), operators can be defined as follows. The response time domain would include the end-to-end delays introduced by the driver's application, PS, mediator, and CS:

$$\delta^{e2e} = \bigwedge(\texttt{lifetime},\ (\delta_{\texttt{publisher\_app}} \oplus \delta_{\texttt{publisher\_mdw}} \oplus \delta_{\texttt{receiver\_mdw}} \oplus \delta_{\texttt{mediator}} \oplus \delta_{\texttt{sender\_mdw}} \oplus \delta_{\texttt{client\_mdw}} \oplus \delta_{\texttt{client\_app}})) \tag{8}$$

Here, the response time is the minimum of the `lifetime` and the summation of delays in message passing. The reliability domain depends on the intermediate link between the driver's app and the car. Here, as the reliable PS is used, the reliability is dependent on the reliability of the CS connection:

$$\mathcal{R}^{e2e} = \bigvee(\mathcal{R}_{\texttt{mediator}}, \mathcal{R}_{\texttt{sender\_mdw}}, \mathcal{R}_{\texttt{client\_mdw}}) \tag{9}$$

Finally, the success rate depends on the intermediary rates:

$$\mathcal{S}^{e2e} = (\mathcal{S}_{\texttt{publisher\_app}} \times \mathcal{S}_{\texttt{publisher\_mdw}} \times \mathcal{S}_{\texttt{receiver\_mdw}} \mathcal{S}_{\texttt{mediator}} \times \mathcal{S}_{\texttt{sender\_mdw}} \times \mathcal{S}_{\texttt{client\_mdw}} \times \mathcal{S}_{\texttt{client\_app}}) \tag{10}$$

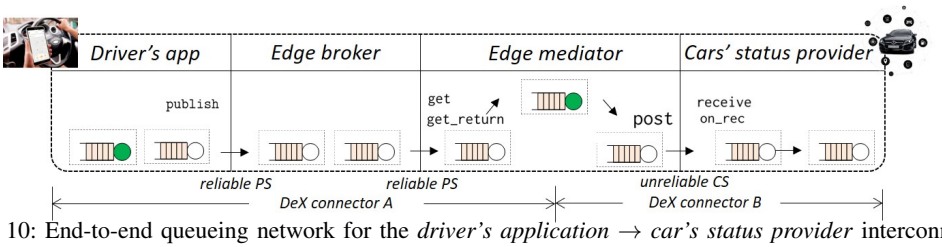


Fig. 10: End-to-end queueing network for the *driver's application* → *car's status provider* interconnection.

While above models are composed by following a "manual process", the synthesizer follows an automated rule-based approach to generate the code simulating interactions in any IoT scenario and scripts for system tuning. In particular, as shown in Fig. 9, the synthesizer takes as input the interface descriptions of the IoT devices. Then, the synthesizer composes: (i) the DeX Mediator; (ii) the end-to-end PerfMP (`e2ePerfM`); and (iii) the end-to-end QoS algebra (`e2eQoSAlgebra`). In our example (Fig. 5), the synthesizer identifies the DeX connectors and their properties (interaction paradigms, QoS delivery mode, lifetime and connection/disconnection periods) based on the interactions between the driver's app and the car's status provider. Initially, the following middleware protocols are identified: (i) the driver's application subscribes to receive notifications using the MQTT "fire-and-forget" delivery mode; and (ii) the car's status provider receives requests using the "non-confirmable" delivery mode. Such information is included in the *Interface Description* (ID) of each IoT device/service.

Based on the extracted CS, PS or DS paradigm, the synthesizer initially defines the sequence of PerfMPs and QoS algebraic operators for "DeX Connector A". For our case (PS), the synthesizer selects the reliable `publish` PerfMP of Fig. 7 (MQTT is QoS_r), that is then parameterized using connection/disconnection periods and the lifetime period. The same applies for the QoS algebraic calculation. The PerfMP of DeX connector A is finally completed by selecting the mediator's reliable PerfMP (`get` part in Fig. 8) and the QoS algebraic operator, by relying on the DeX mapping to CS, PS and DS primitives (defined in Table I). The same process is repeated to define the PerfMPs and QoS algebraic operators of *DeX Connector B*.

Such a composition process can be extended to multiple middleware paradigms and mediators. After both `e2ePerfM` and `e2eQoSAlgebra` are defined, the synthesizer generates the code that mainly consists of a proxy server (DeX mediator, see §III-D), Java queues, Java operators, and code scripts to perform statistical analysis and system tuning (see more details in §VI). A system designer can automatically create such artifacts for any pair of either homogeneous or heterogeneous IoT devices. Queues that represent the mediators' conversion logic, can be parameterized based on the required time for the conversion between specific protocols (e.g., MQTT to CoAP). Do note that the algebraic framework can serve as the baseline in the case of black-box models or multi-vendor deployments, where detailed queueing network modeling is unavailable. The observed metrics may be used to build these composed models.

The synthesizer, the interface descriptions of IoT devices and the code of our running example are provided at https://github.com/satrai-lab/dexms-qos.

VI. End-to-End Performance Evaluation

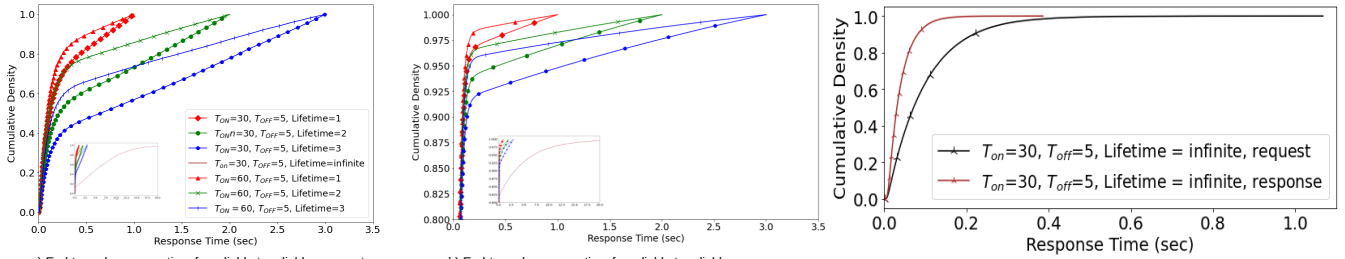
This section illustrates how IoT system designers can leverage the QoS-aware DeXMS synthesizer to evaluate the interoperability effectiveness in the IoV scenario. We then demonstrate how such an evaluation can be used to satisfy Service Level Agreements (SLA) in IoV scenarios. Limitations and threats to validity of our solution are also described.

Experimental Use Case. To evaluate the interactions between different middleware layer protocols, we rely on our running example, the IoV scenario, presented in II and the experimental setup shown in Table III. Cars enter a highway through multiple entry points at an average rate of 42 cars / second [3]. Drivers access digital services provided by road companies, such as traffic services, car diagnostics services, etc. Cars interact with these services by sending request messages through an Edge broker installed on the highway. We consider that every car sends an average number of requests of 3 messages / second. Cars use the MQTT (R) or MQTT-SN (U) protocols at the app-layer, and IEEE 802.11p [24] as the wireless protocol, with a transmission rate of 3 Mbps for the upload link and 6 Mbps for the download link. The Edge broker handles the incoming requests from the cars, converts them to the correct protocol if needed, and forwards the request to the services. We also consider that as cars are travelling on the highway, they experience network disconnections. The cars remain connected for a period T_{ON} before disconnecting from the network for a short time T_{OFF}. The services respond to the cars' requests by sending messages via the Edge broker. However some services (e.g. traffic services, car diagnostics) send notifications periodically to cars for updates. We consider that services send an average number of 6 messages / sec for every car. Also, cars leverage the CoAP protocol to send/received such messages.

Scenario Simulation. The above scenario consists of heterogeneous interactions and requires mapping of both functional (MQTT to CoAP) and QoS semantics ($\texttt{QoS}_\texttt{u}$ to $\texttt{QoS}_\texttt{r}$). As described in §V, the *QoS-aware DeXMS synthesizer* takes as input the interface descriptions of applications and services and provides as output mediators mapping functional semantics and end-to-end Perf and QoS algebra models that map QoS semantics. The model presented in Fig. 7 is given by the synthesizer (as Java code) along with several scripts that

TABLE III: Experimental setup

App Layer Protocols	Wireless Protocol	BW car - edge broker	T_{ON}	T_{OFF}	# incoming cars	$\lambda_{request}$ /car	$\lambda_{request}$ /all cars	$\lambda_{response}$ /car	$\lambda_{response}$ /all cars
MQTT (R) — MQTT-SN(U)	IEEE 802.11p	3 Mbps upload 6 Mbps download	30 sec 60 sec	5 sec	42 cars / sec	3 msg/car/sec	130 msg/sec	6 msg/car/sec	250 msg/sec



a) End-to-end response time for reliable to reliable - request

b) End-to-end response time for reliable to reliable - response

c) End-to-end response time for reliable to unreliable

Fig. 11: End-to-end response times

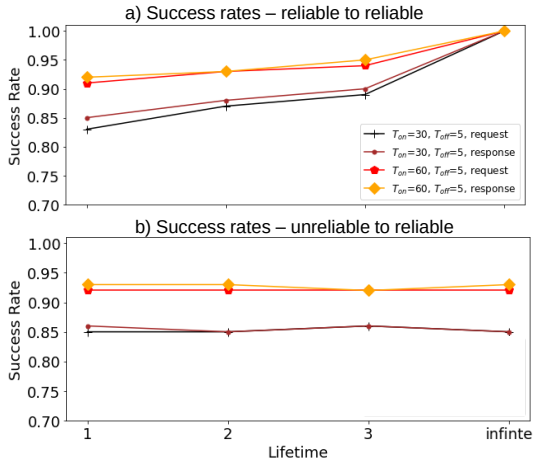


Fig. 12: Success rates

can be used to evaluate the interoperability effectiveness[1]. We consider the two cases where cars use reliable (MQTT) or unreliable (MQTT-SN) protocols. We assume that services use a reliable protocol (i.e., CoAP "confirmable") and that unlike cars, they do not experience network disconnections (i.e., they are always ON). Hence, we create two models: (i) a model for a reliable publisher sending messages to a reliable subscriber, and (ii) a model for an unreliable publisher sending messages to a reliable subscriber.

The synthesizer provides the end-to-end Perf model as Java queues. In particular, it leverages the JINQS [22] library to implement and simulate the composed queueing models. JINQS is a Java simulation library for multiclass queueing networks and it provides a suite of primitives that allow developers to rapidly build simulations for a wide range of QNs. Transmission rates correspond to service rates for queues representing the network resources, while T_{ON} and T_{OFF} periods are modeled as exponentially distributed, with the mean value shown in Table III. Finally, queues have a sufficient buffer capacity so that no messages are dropped (not because of lifetime or unreliable protocols). Using the above settings, the synthesizer runs our simulated models and derives the simulated curves of mean response times and success rates for several configurations as described below. For each configuration (with specific arrival rates, protocols, lifetime, etc.) the queueing model transmits 800,000 messages to collect interaction statistics (cumulative density) and perform statistical analysis. Confidence intervals of the simulation results are found to be very small (less than two orders of magnitude) and are not presented in the figures below.

Results: reliable Publisher. We start by evaluating the performance when the cars use MQTT (R) as the app-layer protocol. We perform experiments by varying the `lifetime` parameter. If a message is not received within the `lifetime` period, it is dropped. Fig. 11a shows the end-to-end response time for request messages sent from cars to services. Setting a higher T_{ON} results in a lower overall response time. For example, for a `lifetime` = 2 seconds and T_{ON} = 60 seconds, 80% of the messages have a lower response time than 1 second, whereas setting T_{ON} = 30 seconds for the same `lifetime` period increases the overall response time by 0.5 seconds for 80% of the messages. This shows that because the publisher is using a reliable protocol, the messages are queued (and not dropped) during disconnection periods. As disconnections become more frequent (T_{ON} = 30 seconds vs. T_{OFF} = 60 seconds), more messages are queued and the queueing time increases, resulting in a higher overall response time. We also notice that setting a longer `lifetime` period increases the response time of messages. This happens because without a `lifetime` the response time will be unbounded. This is reflected in the curve showing the response times with infinite `lifetime`, where the response time increases significantly. However, the delivery success rate is bounded by the `lifetime` periods, as depicted in Fig. 12a. We can see how the success rate increases by 10% with a higher T_{ON}. With an infinite lifetime, the success rate is 100%. This is expected because with a longer `lifetime`, fewer messages expire and are dropped. Without assigning a `lifetime` period, all messages will be delivered. The behavior for response messages is similar; however because the download link has a higher transmission rate than the upload link, we see in Fig. 11b that response messages have lower response times.

Results: unreliable Publisher. Drivers using an unreliable application layer protocol such as MQTT-SN (U) will experience a much lower response time compared to drivers using MQTT (R), as Fig. 11c clearly shows. Even with an infinite lifetime,

[1]Interface descriptions and scripts for this experiment are provided at: https://github.com/satrai-lab/dexms-qos

the response time of all messages is lower than 1 second with a T_{ON} period of 60 seconds. However, this comes at the expense of a lower delivery success rate: because the response time is low, setting a `lifetime` period doesn't significantly affect the success rates of messages. The success rate is 85% with a T_{ON} = 30 seconds, and 91% with a T_{ON} = 60 seconds (Fig. 12b). Notice that in this case, even without assigning a lifetime period to messages, the success rate is not going to be 100%. This comes as a result of using an unreliable protocol, where messages will be dropped during disconnection periods.

Satisfying SLR. 5GAA [7] and 5GPPP [1] define Service Level Requirements (SLR) according to use cases for IoV scenarios. We evaluate the performance of the middleware layer protocols against some of the defined SLRs. For example, 5GPPP specifies that the delay for Vehicle-to-Infrastructure (V2I) interactions should be within 100 ms. According to experiments presented above, this response time is achievable only when drivers use an unreliable (MQTT-SN) protocol. In this case, 80% of the messages will have a lower response time than 100 ms, and hence stay within the bounds of the SLRs. For traffic information services, a delay of 2 seconds is tolerated. Thus, a reliable (MQTT) protocol can be used with a `lifetime` period of 2 seconds, since all messages will be received within the bounds of the SLR. Application and middleware developers need to know what parameters are best suited to satisfy the SLR, depending on the use case of interest. To this end, we develop a performance tuning tool that allows developers to explore the tuning parameters needed according to their needs. By relying on toolkits such as *PerfMP* and *JINQS*, we can analyse multiple IoV scenarios to study the effects of varying parameters on SLRs.

Threats to Validity. Our approach exhibits the following threats to validity. First, while generic PerfMPs for well known interaction paradigms (and middleware protocols) are offered, properties of protocols that are not common within the same interaction paradigm may not be captured. In addition, PerfMPs have been validated against real protocols in the literature [11], however, their applicability in real-world applications is lacking. Another limitation is that when highly heterogeneous arrival rates (sensor data vs. camera feeds) are applied, performance metrics are not accurate (this is similar in bottleneck cases). Furthermore, modeling system properties such as network bandwidth may provide results close to network simulators, however there are deviations.

VII. Related Work

This section explores research on Middleware interoperability. While functional interoperability has garnered significant research interest, there is a lack of emphasis on non-functional interoperability approaches, particularly among heterogeneous IoT devices.

Functional Interoperability: Interoperability at the middleware layer can be approached by various paradigms such as Service Oriented Architectures [19], [31], IoT Gateways [17], [5] and Cloud Computing platforms [32]. In [19], protocol translators are devised that utilize an intermediate format to capture all protocol specific information for heterogeneous interactions. XWARE [31] is an event-based framework for solving interoperability across different middleware platforms by using plugins and mediators. Gateway approaches such as [17] provide interoperability between REST, CoAP, MQTT and XMPP protocols. Cloud platforms include solutions such as H2020 symbIoTe [32], that provides an interoperability framework across vertical IoT platforms for enabling cross-platform application development.

Homogeneous Non-Functional Interoperability: To optimize response times and delivery rates in IoT systems, modeling approaches for middleware protocol performance evaluation must be investigated. In [23], [20], [18] the trade-off between response times and delivery success rates is evaluated for key IoT protocols (e.g., CoAP). However, the employed methods are protocol-specific and do not cover the introduction of new IoT protocols. Several existing efforts concerning the evaluation of mobile systems aim at satisfying QoS requirements under several constraints (e.g., limited resources) [29], [27], [30], [34]. These papers assume homegeneous middleware interactions for the analysis of non-functional properties.

Heterogeneous Non-Functional Interoperability: The authors have found limited research on the non-functional property analysis in middleware connectors. In [12], IoT interactions using the DeX connector model is enhanced to accurately model timing behavior through timed automata. Verification of conditions for successful DeX interactions is done in UPPAAL in conjunction with the timing guards specified. The non-functional interactions are extended to reliability and success rate in this paper through the use of queuing network models and QoS algebraic composition. This analysis is crucial for large scale IoT deployments to ensure timely and successful heterogeneous middleware interactions.

VIII. Conclusions

Large scale deployments of Internet of Things (IoT) applications require heterogeneous interactions between multiple paradigms such as asynchronous messaging. streaming and publish-subscribe messaging. A further requirement is ensuring end-to-end (multi-dimensional) Quality of Service (QoS) spanning multiple paradigms. To meet these objectives, this paper introduces a platform for the automated synthesis of QoS-aware mediating artifacts. Such mediators enable the interconnection between IoT devices employing heterogeneous middleware protocols and evaluation of end-to-end QoS. Formalisms such as queueing network models and QoS algebraic composition are used to study the non-functional properties of heterogeneous middleware connectors. This approach is validated via experimentation with an Internet-of-Vehicles case study employing heterogeneous middleware protocols.

Acknowledgements

This work is partially supported by the Horizon Europe project DI-Hydro under grant agreement number 101122311.

REFERENCES

[1] https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Automotive-Vertical-Sectors.pdf, 5GPPP.

[2] https://cloud.google.com/maps-platform/places, Google Places.

[3] https://www.dir.ile-de-france.developpement-durable.gouv.fr/les-comptages-a174.html, Ile de France Stats.

[4] http://www.oracle.com/technetwork/java/jms/index.html, Sun Microsystems. JMS Specifications and Reference Implementation.

[5] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi. Toward better horizontal integration among IoT services. *IEEE Communications Magazine*, 53(9):72–79, 2015.

[6] L. Aldred, W. van der Aalst, M. Dumas, and A. ter Hofstede. On the notion of coupling in communication middleware. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, Agia Napa, Cyprus, October 2005.

[7] 5GAA Automotive Association et al. White Paper C-V2X use cases, methodology, examples, and service level requirements, 2019.

[8] A. Banks and R. Gupta. MQTT Version 3.1. 1. *OASIS standard*, 2014.

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

[10] A. Benveniste, C. Jard, A. Kattepur, S. Rosario, and J. Thywissen. QoS-Aware Management of Monotonic Service Orchestrations. *Formal Methods in System Design*, 44:1, 02 2014.

[11] G. Bouloukakis, K. Benson, L. Scalzotto, P. Bellavista, C. Grant, V. Issarny, S. Mehrotra, I. Moscholios, and N. Venkatasubramanian. PrioDeX: a Data Exchange middleware for efficient event prioritization in SDN-based IoT systems. *ACM Transactions on Internet of Things*, 2(3):1–32, 2021.

[12] G. Bouloukakis, N. Georgantas, A. Kattepur, and V. Issarny. Timed protocol analysis of interconnected mobile IoT devices. *Journal of Internet Services and Applications*, 12(1):1–31, 2021.

[13] G. Bouloukakis, N. Georgantas, A. Kattepur, and V. Issarny. Timeliness Evaluation of Intermittent Mobile Connectivity over Pub/Sub Systems. In *ACM/SPEC ICPE*, L Aquila, Italy, April 2017.

[14] G. Bouloukakis, N. Georgantas, P. Ntumba, and V. Issarny. Automated Synthesis of Mediators for Middleware-layer Protocol Interoperability in the IoT. *Future Generation Computer Systems*, 101:1271 – 1294, 2019.

[15] G. Bouloukakis, A. Kattepur, N. Georgantas, and V. Issarny. Queueing Network Modeling Patterns for Reliable and Unreliable Publish/Subscribe Protocols. In *15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, New York, USA, 2018.

[16] J. Contreras-Castillo, S. Zeadally, and J.A. Guerrero-Ibañez. Internet of Vehicles: Architecture, protocols, and security. *IEEE Internet of Things Journal*, 5(5):3701–3709, 2017.

[17] Wagner Luís de A. M. Macêdo, Tarcísio da Rocha, and Edward David Moreno. GoThings - An Application-layer Gateway Architecture for the Internet of Things. In *WEBIST*, 2015.

[18] N. De Caro et al. Comparison of two lightweight protocols for smartphone-based sensing. In *IEEE SCVT*, Namur, Belgium, November 2013.

[19] H. Derhamy, J. Eliasson, and J. Delsing. IoT Interoperability—On-Demand and Low Latency Transparent Multiprotocol Translator. *IEEE Internet of Things Journal*, 4(5):1754–1763, 2017.

[20] L. Durkop et al. Performance evaluation of M2M protocols over cellular networks in a lab environment. In *IEEE ICIN*, Paris, France, February 2015.

[21] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 2003.

[22] T. Field. JINQS: An extensible library for simulating multiclass queueing networks, v1. 0 user guide, August 2006.

[23] K. Fysarakis et al. Which IoT protocol? comparing standardized approaches over a common m2m application. WashingtonDC, USA, July 2016.

[24] D. Jiang and L. Delgrossi. IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. In *VTC Spring 2008 - IEEE Vehicular Technology Conference*, pages 2036–2040, 2008.

[25] V. Karagiannis et al. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*, 2015.

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

[27] K. Lee et al. Mobile data offloading: how much can wifi deliver? In *Proceedings of the 6th International COnference*. ACM, 2010.

[28] S. Lee et al. Correlation analysis of MQTT loss and delay according to QoS level. In *IEEE ICOIN*, Bangkok, Thailand, January 2013.

[29] F. Mehmeti and T. Spyropoulos. Performance analysis of "on-the-spot" mobile data offloading. In *IEEE GLOBECOM*, Atlanta, GA USA, December 2013.

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

[31] F.M. Roth, C. Becker, G. Vega, and P. Lalanda. XWARE—a customizable interoperability framework for pervasive computing systems. *Pervasive and mobile computing*, 47:13–30, 2018.

[32] S. Soursos, I. Žarko, P. Zwickl, I. Gojmerac, G. Bianchi, and G. Carrozzo. Towards the cross-domain interoperability of IoT platforms. In *2016 European Conference on Networks and Communications (EuCNC)*, pages 398–402, 2016.

[33] OASIS Standard. Oasis advanced message queuing protocol (amqp) version 1.0., 2012.

[34] 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.