PlanIoT: a framework for adaptive data flow management in IoT-enhanced spaces

Houssam Hajj Hassan, Georgios Bouloukakis, Ajay Kattepur, Denis Conan, Djamel Belaïd

PlanIoT: A Framework for Adaptive Data Flow Management in IoT-enhanced Spaces

Houssam Hajj Hassan*, Georgios Bouloukakis*, Ajay Kattepur†, Denis Conan*, Djamel Belaïd*
*SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France
{houssam.hajj_hassan, georgios.bouloukakis, denis.conan, djamel.belaid}@telecom-sudparis.eu
†Ericsson AI Research, India, ajay.kattepur@ericsson.com

***Abstract*—This paper presents PlanIoT, a middleware approach for enabling adaptive data flow management in IoT-enhanced spaces (e.g., buildings) using automated planning methodologies. Today's sensorized spaces deploy applications falling to diverse categories such as *analytics*, *real-time*, *transactional*, *video streaming* and *emergency response*. Depending on the category, applications have different QoS requirements related to timely delivery, networking resources, accuracy, etc. Typically, state-of-the-art data exchange systems introduce policies for bandwidth allocation or prioritization for specific data types and applications (e.g., camera data). PlanIoT introduces a generic QoS model to evaluate the performance of data flowing in Edge infrastructures and generates their performance metrics dataset. Such a dataset is used as input to automated planning representations to intelligently satisfy QoS requirements of deployed applications. The experimental results show that PlanIoT improves the end-to-end response time of time-sensitive flows by more than 50%, especially with an overloaded Edge infrastructure. We also show the adaptivity of our approach by considering emergency cases that require Edge infrastructure reconfiguration.**

***Index Terms*—Adaptive Systems, Automated Planning, IoT, QoS, Smart Spaces.**

I. Introduction

With the advent of Internet of Things (IoT) devices and supporting technologies, spaces (e.g., buildings, homes) are becoming smarter and interconnected. Edge-based infrastructures of today's sensorized buildings include IoT devices to sense physical phenomena or receive actuation commands; software components to process raw data and provide semantically enriched data; message brokers to exchange data; Edge servers as hosting nodes; and the networking infrastructure. IoT applications operating over such infrastructures provide services (e.g., building utility optimization, air/noise monitoring, enforcing emergency & rescue procedures) to improve people's daily activities, life quality, and public safety.

The developed applications are often composed from different Quality of Service (QoS) requirements (response time, accuracy, allowed loss rates, etc.) to support application types such as: (i) *IoT analytics (AN)* based on sensor readings for behavior analysis; (ii) *real-time (RT)* for data acquisition within a limited latency bound; (iii) *transactional (TS)* for request-response related applications; (iv) *video streaming (VS)* for bandwidth and data-intensive applications; (v) *emergency response (EM)* for handling critical situations as soon as possible. IoT-enhanced spaces usually deploy applications falling to all categories: for instance, an occupancy application that indicates whether certain building zones are highly occupied or not; a real-time self-driving robot delivering mails in a university campus; transactional office-related digital services; a video streaming camera surveillance system in workplaces; and an evacuation planning application for buildings.

Serving applications falling to these types may require a tsunami of data that keeps growing. For instance, an occupancy monitoring application deployed in a large university campus (with over 200 buildings) must process WIFI connectivity data in the order of millions each day [1]. In addition, bandwidth and data-intensive applications such as EM and VS affect the performance of all applications. Furthermore, smart space administrators wish to leverage IoT sources to build multi-purpose applications spanning the above categories. For example, images captured from a camera can be used as input to **both** an application that creates evacuation plans (EM), and to an application for detecting parking violations (VS) [2], [3]. We define such applications as ***Intersecting IoT Applications***.

Deploying applications (including intersecting ones) falling to AN, RT, TS, VS and EM categories requires to address the following research questions: *(R1) How can different categories of IoT applications be expressed with different QoS requirements? (R2) How does an Edge infrastructure of a smart space have to manage data flows sent to IoT applications? (R3) What if several identical flows must be delivered to intersecting applications that belong to all categories?* State-of-the-art messaging systems for smart spaces focus on ensuring application portability [4]–[6], resolving conflicts [7], or message prioritization [8], [9]. While middleware-based approaches provide solutions for manipulating data at both the middleware and network layers [10]–[12], they provide support for application-specific data (e.g., VS) and requirements.

This paper introduces PlanIoT, a framework-based solution that enables adaptive data flow management at the middleware-layer using automated planning methodologies [13]. PlanIoT enables defining application categories using technical specifications such as 3GPP [14]–[16]. A generic QoS model evaluates the performance of data flows (exchanged between applications and devices) using different IoT devices, applications and QoS configurations (network resource allocation, priority policies, etc.). PlanIoT composes multiple instances of these QoS models to provide a *dataset* with metrics related to the applications' QoS requirements (i.e., response times,

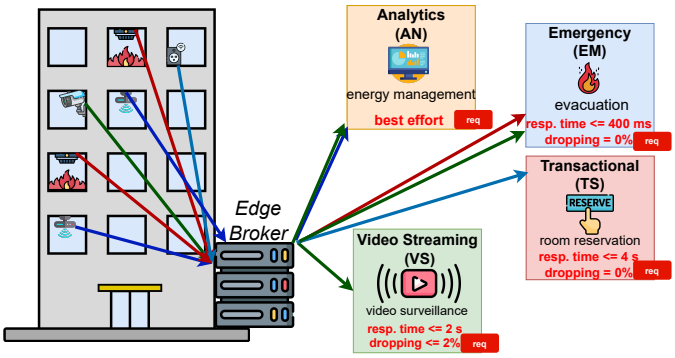

Fig. 1: Smart building scenario

throughput). This dataset is then used to create multiple instances of *domain models* and *problem specifications* that are finally given as input to an AI planner. The planner achieves the following goals: (i) identify adaptive plans to satisfy QoS requirements of IoT applications; and (ii) plan the delivery of identical flows to intersecting applications regardless of their application category. The main contributions of this paper are:

- A middleware-based approach that categorizes applications based on QoS requirements to provide adaptive flow management by relying on QoS modeling and automated planning methodologies (§II).
- A generic QoS model that evaluates the performance of Edge infrastructures using different (and possibly optimized) QoS configurations that capture different smart space situations (§III).
- Generating plans for the adaptive management of data flows to satisfy QoS requirements of (possibly intersecting) applications (§IV).

§V shows how PlanIoT improves the Edge infrastructure's performance and enables runtime adaptation. §VI compares PlanIoT against related works and §VII concludes this paper.

II. The PlanIoT approach

A. Motivating Scenario

We consider the case of a smart office building, as depicted in Fig. 1, that might be equipped with IoT devices such as smoke detectors, temperature sensors, smart plugs, to name a few. An Edge-based IoT platform can be leveraged to forward data from devices to the corresponding IoT applications and services. For instance, an *evacuation planning* application (EM category) can be triggered during an emergency case (fire, earthquake, etc.) to create evacuation plans in a building using data from CCTV cameras and other devices (e.g., smoke, temperature sensors etc.). At the same time, a *video surveillance* application (VS category) used by security officers receives CCTV camera data and shows video footage of different locations within the building. Note that the EM application requires camera data to be received *as soon as possible* (i.e., a transmission time less than 400 ms), whereas the VS application tolerates a higher latency of around 2 seconds. Finally, under normal situations, building occupants may use a *reservation* application that displays the occupancy of meeting rooms and enables their reservation.

Respecting QoS requirements of all applications deployed in IoT-enhanced spaces is not a trivial task, especially for *intersecting applications* (e.g., evacuation planning vs. video surveillance) over a constrained infrastructure. We can observe that applications in such spaces receive heterogeneous data in terms of size, format, urgency and frequency. For instance, a CCTV camera provides multiple data frames per second, usually of considerable size (e.g., 20 KB), whereas a temperature sensor only sends a few bits of data every several seconds. In addition, applications are highly diverse in terms of QoS requirements and data recipients under different circumstances. For instance, during an emergency case, the evacuation planning application holds a higher importance than the video surveillance application due to the safety risks that a potential fire/earthquake may present. In such a case, it is critical to guarantee the timely and reliable delivery of data to the evacuation planning application, and *at the same time* manage the infrastructure so that the QoS requirements of the video surveillance application are not violated. The main challenge here lies in treating the *same data observations* coming from the *same device*, differently, depending on the nature of the receiving application. This task becomes more complex when we consider constrained resources at the Edge.

Existing IoT platforms use all networking resources available to forward the data to IoT applications. Other approaches, such as [8], [9], [17], [18], deal with emergency or real-time applications but are specific to a certain type of applications. Thus, generic and dynamic solutions are required to manage data flows at the Edge.

B. PlanIoT Overview

We now provide an overview of the PlanIoT framework. As shown in Fig. 2, this consists of three main components: (i) ***Edge Infrastructure***; (ii) ***Design Time Synthesis***; and (iii) ***Runtime Automated Planning***. IoT applications and devices deployed in a smart space are part of the *Edge Infrastructure* and they exchange data via a publish/subscribe broker deployed at the Edge. Here, the main challenge is to enable adaptive management of data flowing to IoT applications for satisfying their (possibly diverse) QoS requirements. This is even more challenging when **intersecting IoT applications** belonging to different categories consume common data flows. To enable adaptive flow management, the *Design Time Synthesis* component provides (at design-time) a *dataset* that captures the performance of data flows in an IoT space for different space characteristics and (possibly dynamic) situations. This dataset is then used by the *Runtime Automated Planning* component to perform runtime flow adaptation via the generation of plans that are applied to the *Edge Infrastructure* component. A detailed description of each PlanIoT core component is provided below.

1) Edge Infrastructure: As already pointed out, the Edge infrastructure consists of IoT devices and applications that interact with each other via an adaptive Edge broker (single node or a cluster of distributed broker nodes). ***IoT devices*** act as publishers and produce data encapsulated into messages that

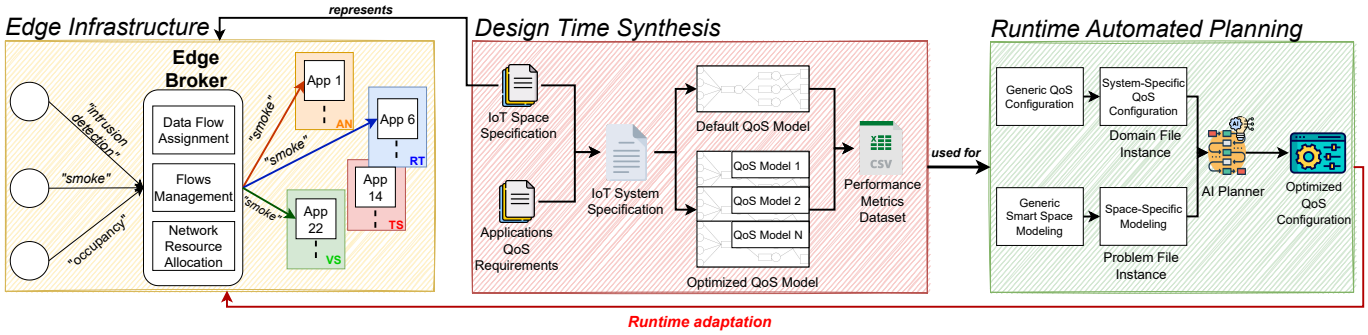

Fig. 2: The PlanIoT Architecture

are tagged with a ***topic*** for routing. ***IoT Applications*** subscribe to receive the produced data using topic-based subscription filters. We group applications based on their category. We also define *intersecting applications* as the set of applications subscribing to receive data with the same topic (e.g., "smoke" data in Fig. 2) regardless of their (possibly) heterogeneous QoS requirements (see §III). To manage data flows such that QoS requirements of all subscribing IoT applications are satisfied, PlanIoT uses the following components. The **Data Flow Assignment** component maps each subscription to a flow and labels it with the corresponding topic, subscribing application, and application category. The **Flows Management** component configures data flows (e.g., via prioritization) based on the optimal QoS configuration received from the automated planning component (see § III-A). Finally, the **Network Resource Allocation** component enables the optimal network allocation policy depending on application categories.

2) Design Time Synthesis: PlanIoT uses the *performance metrics dataset* generated at design time to perform runtime adaptations. In particular, as depicted in Fig. 2, the ***IoT space specification*** and ***Application QoS requirements*** subcomponents are used to compose the ***IoT system specification*** that represents the overall Edge infrastructure of a smart space in a structured way[1]. The specification includes information related to the (i) *IoT devices*: average message sizes, frequency (e.g., periodic, event-driven) as well as the space properties (e.g., temperature of a room); (ii) *IoT applications*: number of applications deployed, their category, their topic-based subscription filters, and their QoS requirements; and the (iii) *Edge broker*: the available network resources and the system capacity. Note that multiple IoT system specifications can be defined for representing multiple situations (e.g., emergency cases).

We rely on existing standards (e.g., 3GPP, ETSI) to define abstract QoS requirements of applications based on the four defined categories (i.e., AN, RT, TS, VS). Such requirements have to be respected for the well-functioning of IoT applications (e.g., response times) under normal conditions. However, using the PlanIoT framework, IoT designers can further define abstract QoS requirements depending on the deployed IoT applications of a space (e.g., applications for emergency response cases). The ***IoT system specification*** is used to automatically create ***QoS Models*** (see §III) that evaluate the performance of data flowing between Edge devices and applications. We define *default* QoS models to evaluate the performance of data exchanged using basic messaging systems (e.g., existing IoT platforms), as well as *optimized* QoS models that consider different QoS configurations (prioritization of applications, dropping of messages) of state-of-the-art IoT platforms that can possibly improve the performance. A ***dataset*** is then generated, which contains metrics (e.g., response times) for the performance of data flows in the Edge infrastructure under the different QoS configurations and smart space parameters (e.g., overloaded Edge infrastructure). More details about the dataset are provided in §III.

3) Runtime Automated Planning: We leverage the performance metrics dataset - which includes multiple and potentially unexpected states of the Edge infrastructure - as input to an ***AI Planner*** (see §IV) that generates plans to dynamically manage data flows. The AI planner takes as input a *domain file* with actions corresponding to the possible QoS models, and a *problem file* describing the Edge infrastructure of the smart space (see §III-C and §IV), as well as the desired goal state (e.g., satisfy QoS requirements of RT). The AI planner uses heuristics and search algorithms to search for actions reaching the goal state (see §III-C). We create *templates* of domain and problem files that can be instantiated at runtime to generate *instances* of domain and problem files. Such instances represent a specific configuration of the Edge infrastructure and can be used for *re-adaptation* purposes at runtime.

III. IoT Flow Handling Problem

This section presents the formal model of the PlanIoT framework. First, we provide an overview of our QoS model that evaluates the performance of data flowing in Edge infrastructures of IoT-enhanced spaces. We then introduce the challenges involved in managing QoS of data flows in Edge infrastructures. Finally, the planning methodology of PlanIoT is formally presented. Refer to Table I for notations used throughout this section.

A. PlanIoT Formal Model

In PlanIoT, data flows are exchanged between IoT devices and applications by following the publish/subscribe interaction paradigm. As depicted in Fig. 3, each IoT device $d_i \in D$ (e.g., smoke detector) acts as a publisher publishing data to

[1]Note that, standard ways of IoT space representation such as Ontologies can be leveraged. However, this is out of the scope of this paper.

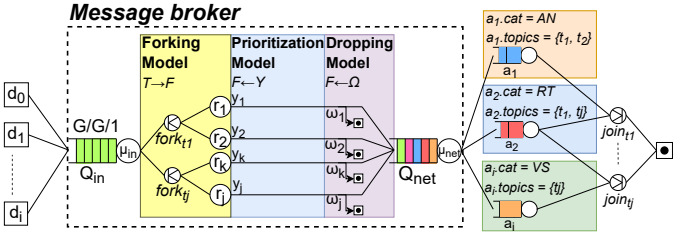

Fig. 3: The PlanIoT QoS Model

Notation	Description
$d_i \in D$; $\mathrm{a}_i \in A$	IoT devices; IoT applications
$t_j \in T$; T_{a_i}	topics; set of a_i topics
$\cap A_i^{t_j}$	intersecting applications
G_{t_j}; λ_{t_j}	topic message size; publication rate
r_j; $f_i \in F$; $F(\cap A_i^{t_j})$	subscription; data flow; identical flows
$y \in Y$; $\omega \in \Omega$	priority; dropping rate
δ_{max}; θ_{min}; ω_{max}	max resp. time; min throughput; max drop rate
μ_{net}; μ_{a_i}	net. queue service rate; app. queue service rate
Δ_{f_i}; Θ_{f_i}; Ω_{f_i}	response time of flow; throughput of flow; dropping of flow

TABLE I: Notations of the parameters in the PlanIoT model

one or more topics $t_j \in T$ (e.g., "smoke") through a message broker (defined as b). Topic messages are characterized by their message size G_{t_j} and the rate λ_{t_j} at which they are produced. λ_{t_j} may generate messages based on a probability distribution (e.g., Exponential, Deterministic) or by relying on real-world traces of existing smart spaces (see §V). We define a topic as a tuple $t_j = (G_{t_j}, \lambda_{t_j})$.

Each application can take the role of a subscriber by subscribing to one or more topics. Let $\mathrm{a}_i \in A$ be the applications table with attributes $\{cat, qos, topics\}$, where cat corresponds to the application's category (AN, RT, TS, VS, EM), qos to the application's QoS requirements, and $topics$ to the set of topics T_{a_i} that the application has subscribed to. We refer to each a_i's attribute (e.g., cat) as $\mathrm{a}_i.cat$. Let $A_{AN} = \{\mathrm{a}_i \in A : \mathrm{a}_i.cat = AN\}$ be the set of applications classified to the analytics category; A_{RT}, A_{TS}, A_{VS}, and A_{EM} are defined similarly. Each application's QoS requirements are defined as a tuple $\mathrm{a}_i.qos = (\delta_{max}, \theta_{min}, \omega_{max})$, where δ_{max} is the maximum tolerated response time, θ_{min} is the minimum required throughput, and ω_{max} is the maximum allowed data drop rate. Let qos_{AN}, qos_{RT}, qos_{TS}, qos_{VS}, and qos_{EM} be the QoS requirements for the corresponding application categories with specific values for the δ, θ and ω parameters (example in §V).

Applications indirectly subscribe to topics by specifying subscription filters. We define a subscription as the tuple $r_j = (\mathrm{a}_i, t_j) \in R$, where application a_i subscribes to the topic t_j. Multiple applications may subscribe to receive data matching the same topic filter. However, their QoS requirements are different and thus, their data flows must accordingly be managed by PlanIoT. Let $\cap A_i^{t_j} \in A$ be the i_{th} set of ***intersecting applications*** that have subscribed to the same topic t_j; e.g., the video surveillance (VS) and evacuation planning (EM) applications (§II-A) where their subscriptions intersect and their category is different. For each subscription r_j, we define a ***data flow*** $f_i \in F$ matching r_j starting from the broker towards an application a_k, such that $r_j = (\mathrm{a}_k, t_j)$. Let $F_{AN} \subseteq F$ be the set of flows sent to analytics applications A_{AN}; F_{RT}, F_{TS}, F_{VS}, and F_{EM} are defined similarly. Note that *intersecting applications* $\cap A_i^{t_j}$ have a set of ***identical flows*** $F(\cap A_i^{t_j})$, i.e., flows filtered based on t_j.

By taking into consideration applications and their categories, PlanIoT enables data management at the level of a flow (a topic that matches a subscription of an application) by allocating network resources, assigning priorities and drop rates (only to loss-tolerant applications or categories) at the publish/subscribe overlay. We denote by $Y = \{y_{AN}, y_{RT}, y_{TS}, y_{VS}, y_{EM}\}$ the set of priority classes; and $\Omega = \{\omega_{AN}, \omega_{RT}, \omega_{TS}, \omega_{VS}, \omega_{EM}\}$ the set of dropping rates that can be assigned to data flows. Note that for two identical flows f_1 and f_2, matching subscriptions $r_1 = (\mathrm{a}_1, t_j)$ and $r_2 = (\mathrm{a}_2, t_j)$, we can assign two different priority classes y_1 and y_2 (or dropping rates Ω_1, Ω_2) depending on the applications' categories, i.e. $\mathrm{a}_1.cat$ and $\mathrm{a}_2.cat$.

B. QoS Model

To evaluate the performance of (possibly identical) data flows (F) in Edge infrastructures, we design a QoS model using *Queueing Networks* [19], [20]. As depicted in Fig. 3, an input queue Q_{in} (of G/G/1 type[2]) accepts publications based on topics from the devices D. Its service time represents the subscription matching process that is applied to identify data recipients based on the available subscriptions r_j. To optimize the end-to-end performance of the data flows in Edge infrastructures, our QoS model introduces the following configurations: (1) ***Forking Model:*** for every topic t_j, one fork $fork_{t_j}$ duplicates data towards every subscribed application and creates the corresponding flows $F(\mathrm{a}_i, t_j)$. This step results in the creation of PlanIoT's data flows (F), including the identical ones; (2) ***Prioritization Model:*** sets priorities to topics, however, PlanIoT assigns priorities to *data flows* including the identical ones. To achieve this, it takes into consideration each application's category to assign priorities in a QoS-aware manner (see §III-C); and (3) ***Dropping Model:*** assigns drop rates for each flow f_i for data to be dropped.

To model the networking infrastructure between the Edge broker and the applications A, a multiclass queue Q_{net} is used. The service rate μ_{net} of Q_{net} represents the available network resources. Let W_{DX} be the available networking resources between the PlanIoT data exchange system and all applications A. Then, μ_{net} is calculated based on W_{DX} and the message size of the corresponding flow that is classified to a topic (G_{t_j}). Q_{net} also acts as a priority queue and prioritizes data flows based on their assigned priorities. Let $K_{Q_{in}}$ and $K_{Q_{net}}$ be the queues' maximum capacity that represents the broker's b overall capacity. Although in this work we model a single broker, this can be easily extended for distributed network of brokers by relying on related existing QoS models [21].

At the IoT application side, each a_i is modelled as a queue q_{a_i} that receives flows from the network queue Q_{net} according to a_i's subscriptions; its service rate μ_{a_i} represents

[2]A G/G/1 queue denotes a single-server FIFO queue with general distributions of inter-arrival and service times.

the processing time of the application. A topic join $join_{t_j}$ is used for every topic fork $fork_{t_j}$ that receives data flows and converts them back to topics. This guarantees that the queueing network remains balanced by joining the jobs split by the forks. Note that the networking infrastructure between IoT devices and the Edge broker is not modeled using queueing networks (i.e., negligible response time for the IoT devices to broker links). This is because IoT devices are usually legacy and thus, management of data flows is not possible. Finally, there are no message losses at the network layer due to network constraints and re-transmissions (message dropping is considered at the application-layer only).

Using PlanIoT's QoS model, the following performance metrics can be calculated. Let Δ_{f_i} be the *end-to-end response time* of flow f_i, which represents the time to deliver a message of a data flow f_i to the subscribed application a_i. Let $\Delta_{F_{AN}}$ be the *average end-to-end response time* of flows F_{AN}; $\Delta_{F_{RT}}$, $\Delta_{F_{TS}}$, $\Delta_{F_{VS}}$, and $\Delta_{F_{EM}}$ are defined similarly. In a similar way, we define $\Theta_{F_{AN}}$, $\Theta_{F_{RT}}$, $\Theta_{F_{TS}}$, $\Theta_{F_{VS}}$, and $\Theta_{F_{EM}}$ to be the delivery success rates for the corresponding application categories.

Dataset Generation. We leverage PlanIoT's QoS model to derive metrics that evaluate the performance of data flows in Edge infrastructures without, or with one or more configuration models (*Forking Model*, *Prioritization Model*, and *Dropping Model* shown in Fig. 3). Such a QoS model is automatically composed by PlanIoT using the *IoT system specification* as input (see Fig. 2). First, it generates metrics for a default configuration (i.e., without using our configuration models) where we use all networking resources W_{DX} to transmit data to the applications without taking into account their QoS requirements. For this configuration, all topics have equal priorities and the data drop rates are null. The resulting metrics are calculated per topic. Then, PlanIoT generates metrics using the *Forking Model*. This repeats the default configuration; however, performance metrics are generated per data flow. Next, it uses both the *Forking Model* and the *Prioritization Model* to assign different priorities to data flows. For each flow f_i, we select a priority $y \in \{0, 1, 2, 3, 4\}$ based on the flow's application category (i.e., y_{AN}). For example, possible assignments of priorities can be: $Y = \{0, 1, 1, 1, 1\}$ or $Y = \{4, 0, 3, 2, 1\}$. Similarly, using the *Dropping Model*, different data drop rates are assigned to different flows. Note that the drop rates assigned to flows are lower than the tolerated drop rates specified in the QoS requirements of the application receiving the flow (i.e., for $f_i \in F_{AN}$, we assign a drop rate $\omega_{f_i} \leq \omega_{AN}$).

PlanIoT automatically generates *per-flow-f_i end-to-end metrics* (from publishers to applications) for response times, throughput, and losses. We group the metrics in a CSV file where each row is a tuple $row_{f_i} = (\Delta_{f_i}, \Theta_{f_i}, \Omega_{f_i})$.

C. Automated Planning

While current messaging systems [8], [9], [22] propose advanced techniques for network resource allocation or prioritization/dropping, there exist a few issues:

(1) All flows f_i belonging to a topic t_j are treated in a homogeneous fashion, irrespective of the application category or recipient application that can be intersecting. This allocation can cause deteriorated service in the case of certain types of applications such as A_{EM}, especially in emergency scenarios. **(2)** The priorities Y are specified in a static manner at design time rather than dynamically set using the application level QoS requirements. In contrast, through PlanIoT, the priorities are composed with dropping rates at design time. Introduction of new requirements at runtime are mapped to application categories to ensure appropriate QoS for high priority services.

For each application with $\mathrm{a}_i.qos$ = (δ_{max}, θ_{min}, ω_{max}), PlanIoT provides an optimal QoS configuration such that: $\Delta_{f_i} <= \delta_{max}$, $\Theta_{f_i} >= \theta_{min}$, and $\Omega_{f_i} <= \omega_{max}$. This task becomes challenging to solve for ***identical flows of intersecting applications***. For two applications a_1 and a_2 belonging to the same set of intersecting applications $\cap A_i^{t_j}$ and where $\mathrm{a}_1.qos \neq \mathrm{a}_2.qos$, PlanIoT provides a QoS configuration that satisfies both $\mathrm{a}_1.qos$ and $\mathrm{a}_2.qos$. This is possible through the PlanIoT configuration models (shown in Fig. 3).

Setting these configurations of an increasing number of application categories, devices, topics, priorities and drop rates is non-trivial. A messaging system must handle the scale of flows and the increasing combinatorial state space search. PlanIoT uses AI planning to find an optimal configuration of smart-space Edge infrastructures via design time/runtime adaptations, by considering application requirements. AI Planning [13], [23] begins with the definition of domains, problems and goals that are to be achieved, which are formally defined.

Definition 1: **Planning Domain.** A planning domain is a state transition system $\Sigma = (S, \mathcal{A}, \gamma, \mathcal{C})$, where:

- S is a finite set of states of the system. These refer to the states of the system under consideration, for instance, if the PlanIoT framework is meeting the QoS requirements or is under violation.
- $\mathcal{A}$ is a set of actions that may be performed by an agent (e.g., a data exchange system, that may be performed by the PlanIoT middleware).
- $\gamma : S \times \mathcal{A} \to S$ is the state transition function. If $\gamma(s, \alpha)$ is defined then action α is applicable to state s, with $\gamma(s, \alpha)$ being the predicted outcome.
- $\mathcal{C} : S \times \mathcal{A} \to [0, \infty)$ is a cost function with the same domain as γ. It can represent a cost function minimizing monetary cost, latency or other parameters.

This model does not include the concept of time or concurrent actions. There is also determinism in the outcome of state s when an action α is performed.

Definition 2: **Planning Problem.** A planning problem is a triple $P = (\Sigma, s_0, G)$ where Σ is a state-transition domain, s_0 is the initial state, and G is a set of ground literal goals.
The goal state G is typically the desired state of the system, for instance meeting all QoS constraints.

Definition 3: **Plan.** A plan is a finite set of actions:

$$\pi = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle \tag{1}$$

where the plan's length $|\pi|$ is n. A plan π is applicable to a state $s_0 \in S$ if there are states $s_1, s_2, \ldots, s_n$ such that $\gamma(s_{i-1}, \alpha_i) = s_i$ for $i = 1, \ldots, n$. In this case, $\gamma(s_0, \alpha_\pi) = s_n$ (with α_π being the last action in plan π). A solution for P is a plan π' such that $\gamma(s_0, \alpha_1) \ldots \gamma(s_m, \alpha'_\pi)$ satisfies G.

To illustrate the automation of configuration settings in Fig. 3, we propose a planning problem. The initial state s_0 in the system includes various flows $f_i \in F$ mapped to applications $a_i \in A$ regardless of the application categories. The initial state s_0 also has priority setting $Y = \{y_{AN}, y_{RT}, y_{TS}, y_{VS}, y_{EM}\} = \{0, 0, 0, 0, 0\}$ and drop rate setting $\Omega = \{\omega_{AN}, \omega_{RT}, \omega_{TS}, \omega_{VS}, \omega_{EM}\} = \{0, 0, 0, 0, 0\}$. The goal state G is mapped to one where the QoS requirements for all flows $f_i \in F$ are met ($\Delta_{f_i} <= \delta_{max}$, $\Theta_{f_i} >= \theta_{min}$, and $\Omega_{f_i} <= \omega_{max}$). The planner generates a plan $\pi = \langle \alpha_1, \alpha_2, \alpha_3 \rangle$ to transition from the initial configuration s_0 to goal configuration G. The first action step α_1 is to convert flows of topics to flows of subscriptions (*Forking Model*). The second action step α_2 sets the appropriate $Y = \{y_{AN}, y_{RT}, y_{TS}, y_{VS}, y_{EM}\}$ (*Prioritization Model*). The last action α_3 sets the appropriate dropping rate $\Omega = \{\omega_{AN}, \omega_{RT}, \omega_{TS}, \omega_{VS}, \omega_{EM}\}$ (*Dropping Model*). Based on the plan output, the following actions could be applied to f_i:

$$\begin{array}{ll} [1] & f_i \mapsto r_j \\ [2] & f_i \mapsto Y = \{y_{AN}, y_{RT}, y_{TS}, y_{VS}, y_{EM}\} \\ [3] & f_i \mapsto \Omega = \{\omega_{AN}, \omega_{RT}, \omega_{TS}, \omega_{VS}, \omega_{EM}\} \end{array} \quad (2)$$

The plan π ensures that the higher priority intersecting applications receive more of the resources, specially in case of resource contention. Techniques used to generate plans from the initial state to the goal state include: (i) graph based search techniques, (ii) state-transition systems, (iii) constraint solvers that make use of symbolic predicates, constraints and effects, (iv) heuristic approximations such as removing negative predicates. Comparison of scale, benchmarking and speed of AI planning solvers has been evaluated within the International Planning Competition held bi-anually since 2000[3]. The planners (classical, temporal, uncertainity tracks) have been compared based on scale of problems, heuristic planning accuracy and time taken to solve benchmark problems.

IV. QoS-aware Planning of IoT Flows

Our approach is based on using AI planning to provide an optimal flow-handling configuration. For this purpose we employ the Planning Domain Definition Language (PDDL) [13], [24], which is an action centered language that provides a standard syntax to describe actions by their parameters, preconditions, and effects. A plan consists of two descriptions: (i) the *domain* description that decouples the parameters of actions from specific objects, initial conditions, and goals, and (ii) the *problem* description that instantiates a grounded problem with objects, initial conditions, goals and cost functions (metrics). The same domain description may be paired with multiple problem instances, with varying grounded objects, initial conditions, goals, and cost functions.

[3]https://www.icaps-conference.org/competitions/

We start by writing the domain definition where we specify the type of objects (Listing 1 Line 1): `Topic` (e.g., "smoke", "occupancy") and `Application` (e.g., fire detection application, occupancy management application). The domain definition file also includes the `predicates` that specify the state of the incoming flows of data (Listing 1 Lines 3–6). We consider that a flow (a tuple of `topic` and `application` pairs) can be under one of the four following states: (i) `default`: with default queue configuration output, (ii) `QoS achieved`: configured to achieve desired QoS, (iii) `priority not set`: priorities of flows have not been set, or (iv) `priority set`: flow priorities set. We further include a function `responsetime` that is used to track the end-to-end response time changes per flow. This function is used as an optimization constraint.

Listing 1: PDDL Domain Model Template

```
1  (:types Topic Application - object))
2  (:predicates
3      (default ?t - Topic ?app - Application)
4      (QoS_achieved ?t - Topic ?app - Application)
5      (priority_not_set ?t - Topic ?app - Application)
6      (priority_set ?t - Topic ?app - Application))
7  (:functions
8      (responsetime ?topic ?app) - number))
9  (:action no_change
10     :parameters (?t  - topic ?app - application)
11     :precondition (and (default ?t  ?app))
12     :effect (and (not (default ?t  ?app))
13         (QoS_achieved?t  ?app) #default_effects#))
14 (:action prioritize_RT_VS_TS_AN
15     :parameters (?t  - topic ?app - application)
16     :precondition (and (priority_not_set ?t  ?app))
17     :effect (and (not (priority_not_set ?t  ?app))
18         (priority_set ?t  ?app) #prioritize_RTVSTSAN#))
19 (:action droppingVS1
20     :parameters (?t  - topic ?app - application)
21     :precondition (and (default ?t  ?app))
22     :effect (and (not (default ?t  ?app))
23         (QoS_achieved ?t  ?app) #droppingVS1_effects#))
```

We further define the actions that can be taken by the AI planner (Listing 1 Lines 9–23). These include an `action` to run the default configuration, another to prioritize flows based on applications (A_{RT}, A_{VS}, A_{TS}, A_{AN}) and one to drop messages. Note that Listing 1 contains only a ***subset*** of the action description with parameters, pre-conditions, post-conditions, and timing constraints. The pre-conditions of these actions include the QoS not being met and the priorities not set. The effects of these actions accurately map the required QoS to flows. These effects are in the form of `(increase (responsetime topic i app j) v)`, where `v` represents the value of the end-to-end response time for the flow under the corresponding QoS model.

The planner searches through the combination of default, priority and dropping actions to compose the optimal output for the combination of flows (using graph, heuristic or constraint search [13]). Note that searching through this configuration of flows involves a large search space and is non-trivial; generating an optimal plan requires accurate inputs from the queuing model, domain modeling involving pre-conditions/effects and efficient solvers to handle large combinatorial state spaces. Using PDDL and associated solvers such as Metric-FF [25], PlanIoT is able to perform efficient search across various filtering, priority and dropping combinations.

In the PDDL problem file, we instantiate objects (topics and applications) and specify the initial state of our system. Initially, all flows of data are configured with the default queueing network model settings (Listing 2 Lines 1–3). Our goal is to transmit all flows of data while minimizing the total time for transmission (Listing 2 Lines 4–7). The `responsetime` function is initialized to 0 with each action providing increments to the `responsetime` function. The AI planner will choose the actions that introduce minimum end-to-end response time for all flows. The goal states for each of the flows include appropriate priority setting and QoS management. The optimization metric focuses on minimizing the weighted end-to-end response time of flows. Note that this is a distinguishing feature of the planner: weights can be provided to prioritize particular flows, which are not straightforward to express within the max-min policy [22] or static priority approaches. Through the use of the `metric minimize` expression in Listing 2 line 6, numeric PDDL solvers (such as Metric-FF [25]) can produce plans that minimize the specified metrics, using techniques such as gradient descent.

Listing 2: PDDL Problem Model Template

```
1 (:init   (default topic_all app_all)
2          (priority_not_set topic_all app_all)
3          (= (responsetime topic_i app_j) 0))
4 (:goal (and (QoS_achieved topic_all app_all)
5          (priority_set topic_all app_all)))
6 (:metric minimize (* 1 (responsetime topic10 app11 ))
7        (* 1 (responsetime topic11 app8 )))
```

A part of an example output using PDDL planners such as Metric-FF [25] is provided below:

```
1 : ff: found legal plan as follows
2 step    0: DROPPINGVS1 TOPIC_ALL APP_ALL
3         1: PRIORITIZE_RT_VS_TS_AN TOPIC_ALL APP_ALL
```

The plan includes setting a specific drop rate for VS flows and appropriate prioritization. Under conditions where there is excessive contention for resources, the plan can drop low priority messages to achieve required QoS for higher priority flows. Note that while we have specified a single grounded problem file for solutions, the novel advantage of including AI planners is to dynamically adapt the system to changes. Here, the PDDL domain file in Listing 1 remains static (with fluent grounding), while the PDDL problem file (Listing 2) is dynamically instantiated with new flows, initialization or requirements. This adaptation ensures that the planning system can generate new plans, without the need for re-modelling the system or individual flows.

Design time adaptations. PlanIoT is a generic and extensible framework for handling data flows in smart spaces. To demonstrate the design-time adaptivity of PlanIoT, we show the steps to be performed for adding a new application category. We consider that a new application category - emergency response (EM) - has been defined by the smart space administrators, to respond to fire/accident emergencies. To define EM applications in the PlanIoT framework, we perform the following process: (1) we introduce the new application category and its requirements in the *IoT system specification* file as follows:

```
1     { "applicationCategory": "EM",
2      "qos_responseTime": 0.25,
3      "qos_throughput": 400,
4      "qos_dropRate": 0 }
```

(2) we generate the corresponding dataset for an emergency scenario using PlanIoT; (3) we instantiate the domain and problem files from the templates corresponding to an emergency situation. These templates contain actions that are only used in such situations. For example, because there is an emergency scenario, we introduce an action for dropping a higher percentage (20%) of AN and VS flows:

```
1     (:action droppingVS20AN20
2     :parameters (?t  - topic ?app - application )
3     :precondition (and (default ?t  ?app ))
4     :effect (and (not (default ?t  ?app ))
5     (QoS_achieved ?t  ?app ) #dropVS20AN20_effects#))
```

Note that the template contains several actions with varying dropping percentages; the planner will then choose the action that satisfies the QoS requirements of the applications; (4) in case of an emergency situtation, we run the AI planner at runtime and generate the optimal QoS model for the broker.

V. Experimental Results

We first evaluate the performance of PlanIoT against a default approach that represents basic IoT platforms, and compare the PlanIoT approach with existing state-of-the-art solutions that deal with the IoT flow handling problem. We then assess the scalability of PlanIoT using a realistic smart-spaces dataset under an increasing number of subscriptions that eventually overload the PlanIoT broker. Finally, we show how PlanIoT enables edge infrastructure reconfiguration in the event of an emergency. The PlanIoT code to compose queueing networks, datasets and the plans for an Edge infrastructure is provided at https://gitlab.com/planiot/planiot-seams2023.

A. Experimental Setup

We rely on the JMT (Java Modelling Tools) simulator [20], [26] to create the queueing network presented in Fig. 3. We create a JAVA-based parser that reads our JSON IoT system representation and automatically generates the corresponding QoS model as JMT queueing networks. We run simulations for different QoS models by changing the *Prioritization Model* and *Dropping Model* presented in §III. We then extract the results of the simulations from the JMT XML files and create a dataset of performance metrics consisting of CSV files, as described in §III-B. Planning techniques are used to find the optimal QoS configuration of PlanIoT. We use PDDL [24] to write the domain and problem files used as an input to the Metric-FF planner [25]. As explained in §IV, we rely on *template* domain and problem files that we then instantiate by reading from the performance metrics dataset and plugging in the needed values for the PDDL actions. To generate numeric plans, we make use of the Metric-FF [25] solver that is extended to handle multiple fluents, constants, actions and numeric expressions. Metric-FF is an extension of the fast-forward planning system that supports reasoning with Numerics. As specified in [27], Metric-FF passes most

PlanIoT System Properties					QoS Requirements		
	$\|T\|$	$\|R\|$	$\Sigma\lambda_{t_j}$ (MB/s)	W_{DX} (MB/s)	δ_{max}	θ_{min}	ω_{max}
AN	15	21	18.5	230	best effort	best effort	best effort
RT	18	21	31.5		<400 ms	384 KB/s	0%
TS	11	18	16		<4 s	-	0%
VS	16	20	55.4		<2 s	384 KB/s	<2%
Total	30	80	121.4	230			

TABLE II: Experimental setup

benchmarks and is able to solve problems at scale within a few seconds. This may be improved via factored planning approaches, as well.

B. Evaluation of the PlanIoT approach

We evaluate the PlanIoT approach using the setup presented in Table II. We consider a medium-loaded PlanIoT broker that accepts publications to 30 topics that provide a load of 121.4 Mb/s. We assume that the bandwidth between devices and the broker is large and sufficient to transmit all data, and hence the transmission delay is negligible. We also consider that the broker has a high processing power that does not introduce significant processing delay. Topic data is generated based on exponential and deterministic distributions with a rate of λ_{t_j}. 16 applications subscribe to PlanIoT to receive data matching the 30 topics, with a total number of 80 subscriptions ($|R|$). Table II also shows the QoS requirements per application category. We create these requirements by relying on the ETSI TS 1212 105 V15.0.0 standard [28].

We start first by validating our approach by comparing PlanIoT with an approach that represents basic IoT platforms. Recall from §III-B that the default QoS model uses all network resources W_{DX} to transmit data flows from the message broker to the subscribers regardless of the application categories. In order to satisfy the QoS requirements of all applications, PlanIoT assigns the following priority classes to the application categories: y_{AN} = 3, y_{RT} = 0, y_{TS} = 2, y_{VS} = 1. In addition, a data drop rate of 1% is applied to VS flows. The priorities and drop rates are decided by relying on the generated dataset and the applications' QoS requirements, as described in §IV. Fig. 4 shows the average end-to-end response time per application category for the default approach vs. the PlanIoT approach. We can see that PlanIoT manages to significantly reduce the end-to-end response time $\Delta_{F_{RT}}$ for RT applications from 0.8 seconds to 0.4 seconds as to satisfy the QoS requirements for RT applications. There is also an improvement for $\Delta_{F_{VS}}$ for VS applications. These improvements come at the expense of having higher end-to-end response times for AN (1.2 seconds) and TS (0.8 seconds) applications. However, because $\Delta_{F_{AN}}$ and $\Delta_{F_{TS}}$ stay under the limits of the QoS requirements, we deem this increase in response time acceptable. To highlight how PlanIoT deals with ***intersecting applications*** (defined in §III-A), we plot in Fig. 5 the end-to-end response time *per topic* (for topics shared by multiple application categories) for both the default model and PlanIoT. We observe that under the default QoS model, for the same topic, the end-to-end response time is the same for all subscribers regardless of their application category and their QoS requirements. However, this is different with PlanIoT, as Fig. 5 clearly shows. PlanIoT manages to take into account the different application categories. Therefore, for the same topic, we achieve to have different values for the end-to-end response time, depending on the requirements of the subscribing application. For instance, for *topic17*, $\Delta_{F_{RT}}$ is 0.4 seconds, $\Delta_{F_{VS}}$ is 0.7 seconds, and $\Delta_{F_{TS}}$ is 1.2 seconds. This shows the effectiveness of assigning priorities per subscription that PlanIoT adopts. Note that PlanIoT is flexible enough to introduce trade-off constraints for balancing resources between all application categories, depending on the QoS requirements defined.

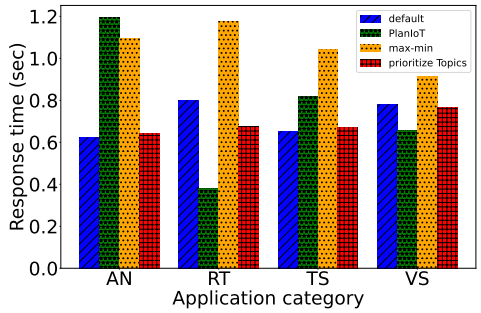

Fig. 4: Comparison of the average end-to-end response time between PlanIoT and existing approaches

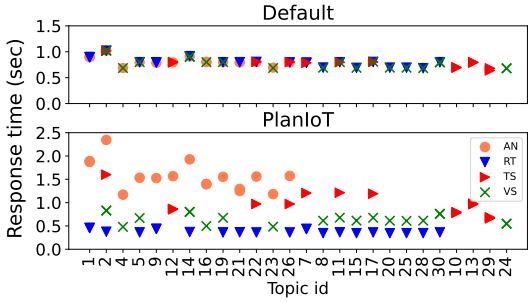

Fig. 5: Response time per topic - Default vs. PlanIoT

C. Comparing PlanIoT with existing solutions

We now compare the PlanIoT approach with two existing state-of-the-art solutions that deal with the flow handling problem, using the same setup of §V-B. We consider the max-min fairness policy [22] used to allocate network resources to the four application categories. The max-min algorithm provides fairness among application categories especially in the presence of network-intensive applications. We also compare our solution with works similar to [9] that prioritize topics belonging to particular applications only. Again, as Fig. 4 shows, we can see that for the application categories with the stricter QoS requirements (RT and VS), PlanIoT has the lowest end-to-end response time among other approaches. Because PlanIoT intelligently assigns priorities and dropping rates *per flow* (as opposed to topics) according to the QoS requirements, we notice that even though the end-to-end response time for AN and TS applications is higher than other approaches, the QoS requirements are still satisfied. Notice how when priorities are applied to topics only, the response time for the four application categories is close. This is because if two or more intersecting applications (from different categories) subscribe to the same topic, they will receive data at the

t_j	topic id	$\|d_i\|$	app. categories
amazon_echo_controller	1	13	RT, TS
building_management_system	2	300	AN, RT, TS
energy_management	3	200	AN, RT, TS
fire_detection	4	100	AN, RT
intrusion_detection	5	50	AN, RT, VS
occupancy_management	6	16	AN, RT, TS, VS
printing	7	20	AN, TS
smart_things_controller	8	12	RT, TS
video_surveillance	9	15	AN, RT, VS

TABLE III: Experimental setup using real traces

# subscriptions	20	40	60	80	100
Default	100%	75%	71.67%	35%	35%
Topic priorities	100%	82.5%	66.67%	35%	48%
Max-min [22]	100%	75%	71.65%	35%	35%
PlanIoT	100%	100%	100%	100%	100%

TABLE IV: Percentage of QoS requirements satisfied

same time. However, when we prioritize subscriptions, we manage to have a different response time for each application, according to its QoS requirements. With the same available resources, PlanIoT manages to satisfy the QoS requirements of all applications, whereas existing solutions fail to satisfy the requirements for RT applications.

D. Scaling up PlanIoT in a smart space

We evaluate the scalability of PlanIoT by testing it under a realistic smart space setup where we increase the number of subscriptions from 20 to 100 to saturate the PlanIoT broker. We rely on the works presented in [29], [30] to create a setup that depicts IoT devices and applications in a smart office building similar to the one presented in Fig. 1. The setup parameters are presented in Table III. We plot in Fig. 6 the variation in the end-to-end response time for each application category as the number of subscriptions increase. The shaded red region in these graphs shows where the utilization of the system is higher than 100%, i.e. the network resources needed to transmit all the flows from the broker to the applications are higher than the total network resources available.

Again, we notice that, by intelligently assigning priorities to application categories and applying drop rates to flows, PlanIoT manages to satisfy the QoS requirements of all applications. In addition, our approach also provides a significantly better performance in terms of the end-to-end response time than other approaches for RT, TS, and VS applications. For RT applications, even with a low-loaded system, the requirements of applications are only satisfied using the PlanIoT approach, which keeps $\Delta_{F_{RT}}$ under the 400 ms limit. For TS and VS applications, other approaches fail to satisfy the requirements as soon as the system becomes overloaded. On the other hand, PlanIoT manages to keep $\Delta_{F_{TS}}$ and $\Delta_{F_{VS}}$ under the requirements' limits. Fig. 7 shows the variation of the throughput as the number of subscriptions increase. We notice that the throughput for PlanIoT is slightly lower than other approaches, especially as the system gets more saturated. This is because of the drop rates Ω that we assign in order to keep the system robust and deliver the most time-sensitive data in a timely manner. However, the throughput for all application categories is higher than the QoS requirements. Table IV presents the percentages of applications that have their QoS requirements satisfied as the number of subscriptions increases. The PlanIoT approach manages to always satisfy the QoS requirements of all applications, even with limited resources.

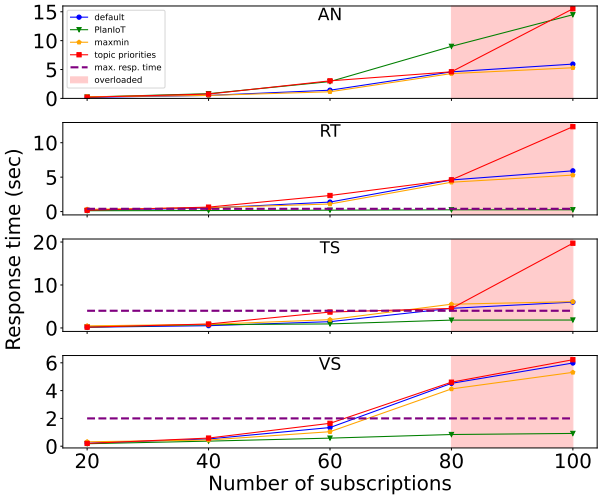

Fig. 6: Response time vs. number of subscriptions

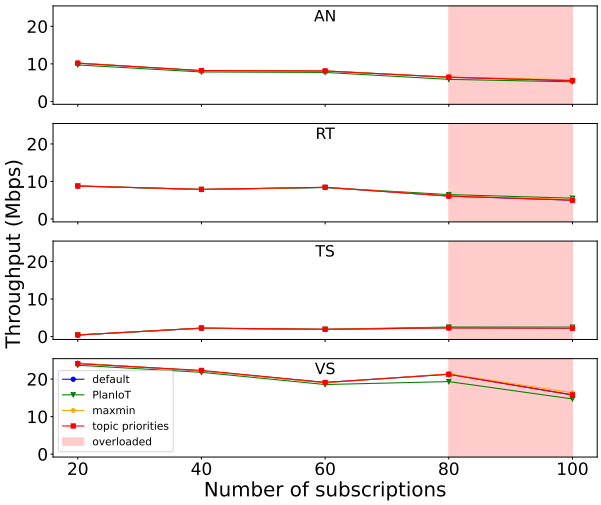

Fig. 7: Throughput vs. number of subscriptions

E. Adding a new application category: Emergency Response

To demonstrate the runtime adaptivity of PlanIoT, we now consider that a fire occurs in the realistic smart space scenario presented in §V-D, where we add 10 subscriptions to the overloaded PlanIoT broker with 80 subscriptions to account for EM applications. Firefighters that come to extinguish the fire will bring devices that may connect to the building's IoT platform and receive data flowing in the building's Edge infrastructure to have a better situational awareness of what is happening in the building. This implies that (i) we now have more publishers and subscribers connected to PlanIoT, and hence more (possibly identical) data flows, and (ii) we have to introduce a new application category (i.e., EM) that has the highest priority among all other applications. The goal here is to (i) satisfy the QoS requirements of EM applications, (ii) keep the PlanIoT broker robust, and (iii) try to satisfy the QoS requirements of the most important applications (RT). We

consider that the user has been proactive and has taken into account the emergency case when generating the performance metrics dataset at design-time, as described in § IV.

Fig. 8 shows that PlanIoT manages to have the lowest end-to-end response time for EM applications (0.2 seconds) in an emergency situation, while at the same time keeping a low end-to-end response time of 0.42 seconds for RT applications. The scatter plot on the right of Fig. 8 shows that for the same topic, we achieve to have the lowest end-to-end response times for subscribers that are EM and RT applications. This shows how in critical situations, PlanIoT manages to treat data flows depending on their QoS requirements, therefore making better use of the available resources to guarantee the timely delivery of the most important data. Nonetheless, notice how the response time for other application categories is high (e.g., 6 seconds for VS). Although PlanIoT fails to manage to have all QoS requirements satisfies, it succeeds in providing a robust performance for the most critical applications.

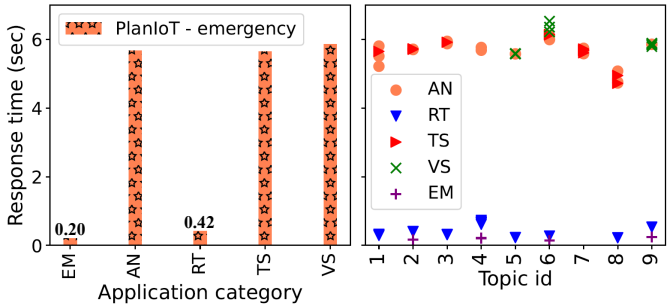

Fig. 8: Average response times in an emergency scenario

VI. Related Work

This section provides an overview of works related to supporting QoS-aware data exchange in smart environments, middleware for IoT and automated planning applications.

IoT-enhanced Spaces. Research related to IoT-enhanced spaces focuses on the heterogeneity of data, avoiding conflict when exchanging data, portability for application deployment across spaces, and prioritizing mission-critical applications. Semantic Web approaches such as *Brick* [4], present a uniform schema for representing metadata in smart buildings. *sTube+* [31] is concerned with the communication between IoT devices deployed in a smart environment and the Cloud via a layered architecture. *RemedIoT* [7] is a dynamic and context-based mediator for IoT event services that deals with the detection and resolution of conflicts when exchanging data. PrioDeX [8], [9] proposes a middleware architecture for smart buildings that enables timely and reliable delivery of the most critical data to relevant data recipients despite challenging network conditions. Unlike these works, the PlanIoT framework considers the diverse requirements and categories of applications that are deployed in IoT-enhanced spaces.

QoS-aware Systems. To enable efficient data exchange in smart environments, existing solutions manipulate data at both the middleware and network layers. Early middleware-based solutions such as *ControlWare* [32] support prioritization or bandwidth allocation based on the available system capacity, data relevance and importance. In [33], a policy-based approach is proposed to manage the QoS related to Internet traffic between tenants in smart buildings. Other solutions assign priorities based on the validity span of data and subscriptions [10] or based on delay and reliability requirements [11]. With the advent of novel networking technologies (e.g. OpenFlow [34]), advanced capabilities are provided to system designers to customize the underlying network infrastructure. SDN-based approaches have been leveraged for priority assignment and bandwidth allocation such as in [35] and [11] that use buffer sharing and prioritization in SDN switches. More recently, machine learning-based solutions [36] are being used to deal with QoS issues. However, such works focus only on the network aspect of QoS and fail to consider the multiple categories of QoS requirements.

Automated Planning. AI Planning is a model-based technology devoted to decision making, which can be used in a variety of application domains [37]. *Roboplanner* [38] is a framework that allows monitoring, state tracking and replanning/configuration of robotic plans using planning techniques. In other application domains, such as [39] and [40], AI planning is adopted to design a Web services composition system. In [41], a MAPE-K loop is introduced that makes use of planning for IoT orchestrations. In [42], Hierarchical Task Networks are used to plan composition of IoT services to meet goal requirements. However, in these related studies, there is no use of explicit data flows, network congestion and scalability effects. To the best of our knowledge, this is the first work that applies AI planning to a middleware system for handling data flows in IoT-enhanced spaces.

VII. Conclusions and Future Work

This paper introduces PlanIoT, a framework for QoS-aware delivery of IoT data flows using automated planning. We propose a generic QoS model to generate a dataset of performance metrics under different configurations. Automated planning techniques are used to optimally configure the Edge infrastructure of smart spaces. Experimental results show that PlanIoT improves the end-to-end response time of time-sensitive applications by more than 50%, and manages to handle diverse applications by assigning priorities and dropping rates based on their QoS requirements. We also demonstrate how PlanIoT can be used for Edge infrastructure re-adaptation by considering an emergency situation.

Our future work includes monitoring the Edge infrastructure for collecting real performance metrics that can be used by the AI planner. We shall also investigate machine learning techniques to predict the performance of the IoT system in unexpected situations. We intend to extend the approach to take into consideration multiple IoT platforms that may exist in a smart community for instance. This presents the challenges of having to handle different data flows among many distributed message brokers, especially when we consider applications that share the same data flows.

REFERENCES

[1] Y. Lin, D. Jiang, R. Yus, G. Bouloukakis, A. Chio, S. Mehrotra, and N. Venkatasubramanian, "Locater: Cleaning Wifi Connectivity Datasets for Semantic Localization," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 329–341, Nov. 2020.

[2] S. Chaturvedi, S. Tyagi, and Y. Simmhan, "Cost-Effective Sharing of Streaming Dataflows for IoT Applications," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1391–1407, 2021.

[3] Q. H. Cao, I. Khan, R. Farahbakhsh, G. Madhusudan, G. M. Lee, and N. Crespi, "A trust model for data sharing in smart cities," in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–7.

[4] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M. Kjærgaard, M. Srivastava, and K. Whitehouse, "Brick: Towards a Unified Metadata Schema For Buildings," in *ACM BuildSys'16*, 2016, pp. 41–50.

[5] R. Yus, G. Bouloukakis, S. Mehrotra, and N. Venkatasubramanian, "Abstracting interactions with iot devices towards a semantic vision of smart spaces," in *6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (Buildsys)*, New York, USA, 2019.

[6] ——, "The semiotic ecosystem: A semantic bridge between iot devices and smart spaces," *ACM Transactions on Internet Technology – TOIT*, 2022.

[7] R. Liu, Z. Wang, L. Garcia, and M. Srivastava, "RemedioT: Remedial Actions for Internet-of-Things Conflicts," in *ACM BuildSys'19*, 2019, pp. 101–110.

[8] 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 Trans. Internet Things*, 2021.

[9] K. Benson, G. Bouloukakis, C. Grant, V. Issarny, S. Mehrotra, I. Moscholios, and N. Venkatasubramanian, "FireDeX: a Prioritized IoT Data Exchange Middleware for Emergency Response," in *ACM/IFIP/USENIX Middleware'2018*, 2018, pp. 279–292.

[10] M. Saghian and R. Ravanmehr, "Publish/Subscribe Middleware for Resource Discovery in MANET," in *IEEE/ACM CCGrid'2015*, 2015, pp. 1205–1208.

[11] Y. Wang, Y. Zhang, and J. Chen, "Pursuing Differentiated Services in a SDN-Based IoT-Oriented Pub/Sub System," in *IEEE ICWS'2017*, jun 2017, pp. 906–909.

[12] S. Khare, H. Sun, K. Zhang, J. Gascon-Samson, A. Gokhale, X. Koutsoukos, and H. Abdelaziz, "Scalable edge computing for low latency data dissemination in topic-based publish/subscribe," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 214–227.

[13] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, 2004.

[14] 3GPP, Technical Specification Group Services and System Aspects, "Service requirements for cyber-physical control applications in vertical domains," 3GPP TS 22.104 (V17.2.0), Dec. 2019.

[15] ——, "Study on Communication for Automation in Vertical Domains (CAV)," 3GPP TR 22.804 (V16.2.0), Dec. 2018.

[16] ——, "System architecture for the 5G System (5GS)," 3GPP TS 23.501 (V16.6.0), Oct. 2020.

[17] K. F. Firdaus, S. A. Wibowo, and K. Anwar, "Multiple access technique for iot networks serving prioritized emergency applications," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019, pp. 1–5.

[18] D. Calvaresi, M. Marinoni, A. Sturm, M. Schumacher, and G. Buttazzo, "The challenge of real-time multi-agent systems for enabling iot and cps," in *Proc. of the International Conference on Web Intelligence*, ser. WI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 356–364.

[19] D. Gross, J. Shortle, J. Thompson, and C. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 4th edition, 2008.

[20] M. Bertoli, G. Casale, and G. Serazzi, "Jmt: performance engineering tools for system modeling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 10–15, 2009.

[21] P. Bellavista, A. Corradi, and A. Reale, "Quality of service in wide scale publish—subscribe systems," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1591–1616, 2014.

[22] D. Pan and Y. Yang, "Max-min fair bandwidth allocation algorithms for packet switches," in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1–10.

[23] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.

[24] M. Fox and D. Long, "PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[25] J. Hoffmann, "Extending ff to numerical state variables," in *ECAI*, vol. 2. Citeseer, 2002, pp. 571–575.

[26] M. Bertoli, G. Casale, and G. Serazzi, "An overview of the jmt queueing network simulator," *Politecnico di Milano-DEI, Tech. Rep. TR*, vol. 2007, 2007.

[27] A. Gerevini, A. Saetti, and I. Serina, "Planning with numerical expressions in lpg," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ser. ECAI'04. NLD: IOS Press, 2004, p. 667–671.

[28] European Telecommunications Standards Institute, "Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; Services and service capabilities," 3GPP TS 22.105 V15.0.0, Jul. 2018.

[29] R. Kumar, M. Swarnkar, G. Singal, and N. Kumar, "Iot network traffic classification using machine learning algorithms: An experimental analysis," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 989–1008, 2022.

[30] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying iot traffic in smart cities and campuses," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2017, pp. 559–564.

[31] C. Hu, W. Bao, D. Wang, Y. Qian, M. Zheng, and S. Wang, "STube+: An IoT Communication Sharing Architecture for Smart After-Sales Maintenance in Buildings," *ACM Trans. Sen. Netw.*, vol. 14, no. 3–4, Nov. 2018.

[32] R. Zhang, C. Lu, T. Abdelzaher, and J. Stankovic, "ControlWare: a middleware architecture for feedback control of software performance," in *IEEE ICDCS'2002*, 2002, pp. 301–310.

[33] F. Martinelli, C. Michailidou, P. Mori, and A. Saracino, "Managing qos in smart buildings through software defined network and usage control," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2019, pp. 626–632.

[34] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[35] D. Singh, B. Ng, Y.-C. Lai, Y.-D. Lin, and W. Seah, "Modelling Software-Defined Networking: Switch Design with Finite Buffer and Priority Queueing," in *IEEE LCN'2017*, 2017, pp. 567–570.

[36] G. White, A. Palade, C. Cabrera, and S. Clarke, "Iotpredict: Collaborative qos prediction in iot," in *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2018, pp. 1–10.

[37] A. Gerevini and I. Serina, "LPG: A Planner Based on Local Search for Planning Graphs with Action Costs," in *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems*, Apr. 2002, pp. 13–22.

[38] A. Kattepur and B. Purushothaman, "RoboPlanner: A Pragmatic Task Planning Framework for Autonomous Robots," *Cognitive Computation and Systems*, vol. 2, Feb. 2020.

[39] S. Qi, X. Tang, and D. Chen, "An Automated Web Services Composition System Based on Service Classification and AI Planning," in *IEEE CGC'2012*, 2012, pp. 537–540.

[40] G. Zou, Y. Chen, Y. Yang, R. Huang, and Y. Xu, "AI planning and combinatorial optimization for web service composition in cloud computing," in *Proc. of the Annual International Conference on Cloud Computing and Virtualization*, 2010, p. 28.

[41] U. Bellur, N. Narendra, and S. Mohalik, "AUSOM: Autonomic Service-Oriented Middleware for IoT-Based Systems," in *IEEE SERVICES'2017*, 2017, pp. 102–105.

[42] J. Bidot, C. Goumopoulos, and I. Calemis, "Using ai planning and late binding for managing service workflows in intelligent environments," in *IEEE PerCom'2011*, 2011, pp. 156–163.