Resilient Edge: Building an adaptive and resilient multi-communication network for IoT Edge using LPWAN and WiFi

Vijay Kumar, Poonam Yadav, *member, IEEE,* and Leandro Soares Indrusiak, *senior member, IEEE*

***Abstract*—Edge computing has gained attention in recent years due to the adoption of many Internet of Things (IoT) applications in domestic, industrial and wild settings. The resiliency and reliability requirements of these applications vary from non-critical (best delivery efforts) to safety-critical with time-bounded guarantees. The network connectivity of IoT edge devices remains the central critical component that needs to meet the time-bounded Quality of Service (QoS) and fault-tolerance guarantees of the applications. Therefore, in this work, we systematically investigate how to meet IoT applications mixed-criticality QoS requirements in multi-communication networks. We (i) present the network resiliency requirements of IoT applications by defining a system model (ii) analyse and evaluate the bandwidth, latency, throughput, maximum packet size of many state-of-the-art LPWAN technologies, such as Sigfox, LoRa, and LTE (CAT-M1/NB-IoT) and Wi-Fi, (iii) implement and evaluate an adaptive system *Resilient Edge* and Criticality-Aware Best Fit (CABF) resource allocation algorithm to meet the application resiliency requirements using Raspberry Pi 4 and Pycom FiPy development board having five multi-communication networks. We present our findings on how to achieve 100% of the best-effort high criticality level message delivery using multi-communication networks.**

***Index Terms*—Internet of Things (IoT), Wireless Networks, Resiliency, Quality of Service (QoS), Low Power Wide Area Networks, Wifi.**

I. Introduction

IOT devices are everywhere sensing, collecting data and providing information to make a better-informed decision about the environment. Many safety-critical IoT applications such as self-health monitoring through wearable IoT devices connect to a mobile phone/local hub via Bluetooth, ZigBee or Wi-Fi and further send the data to a cloud service or hospital central processing system through Internet [1], [2].

In the event of a network-failure, e.g., power outage or any other incidental connection failures, the Wi-Fi could be disconnected temporarily, resulting in either data loss or delayed data communication [3]. Depending on the time of the day, it may take from one minute to several minutes to regain connectivity to the Wi-Fi. On the other hand, according to a survey [4], the average amount of broadband downtime per year in the UK ranges from 25.4 hours to 168.9 hours.

However, for safety-critical applications, it is essential to maintain resilient data connectivity at all time for the delivery of a time-critical message. The LPWAN technologies have explicitly been designed to meet IoT application requirements. They are built on existing cellular systems to provide improved battery life, power efficiency and indoor and outdoor coverage area [5] at an affordable cost. Availability of alternate low power long-range network mediums at a meagre cost opens a new horizon of opportunities.

However, LPWAN technologies also have challenges in terms of limited bandwidth; the number of messages allowed per day and payload size. On the other hand, IoT edge application requirements are defined in terms of message criticality (such as high/low priority), privacy settings, message data length, message sending frequency and user trust on a particular network. Based on the application requirements and available network medium, application traffic can be routed through a specific network medium. Further, in case of a particular network medium unavailability or failure, the application can be informed of the network state and can decide on the suitability of the network and adapt accordingly. For instance, assuming the application is sending data over Wi-Fi and because of power failure Wi-Fi is disconnected, the application can choose to send data over Long-Term Evolution (LTE)(LTE for Machines (LTE-M)/NarrowBand-IoT (NB-IoT)), LoRa (Long Range), Sigfox and adapt parameters such as payload size and frequency accordingly. Despite all the hype and hope of LPWAN, it is not fully understood that if we can achieve network resiliency at the Edge using LPWAN and Wi-Fi for time-critical IoT applications [6], [3], [7]. Therefore in this work, we propose a hypothesis that using LPWAN technologies and Wi-Fi, we can achieve network resiliency at the edge IoT device by providing a capability to choose a suitable network medium based on the application requirements. For the implementation, we utilise affordable, readily-available MicroPython enabled, multi-network micro-controller Pycom FiPy board [8] providing connectivity to Bluetooth, Wi-Fi, LoRa, LTE (CAT-M1/NB-IoT) and Sigfox.

Contributions:

- We present use cases for resiliency requirements of the IoT edge networks;
- provide a detailed analysis of many state-of-the-art LPWAN technologies, such as Sigfox, LoRa, LTE (CAT-M1/NB-IoT) and evaluate their bandwidth, latency, throughput and maximum packet size using an experiment;
- identify and compare resource management approaches

V. Kumar is with the Department of Civil Engineering, University of Bristol, UK, e-mail: vijay.kumar@bristol.ac.uk.

P. Yadav and L.S. Indrusiak are with Computer Science Department, University of York, UK, e-mail: poonam.yadav@york.ac.uk and leandro.indrusiak@york.ac.uk, respectively.

that consider QoS requirements at multiple levels of criticality;
- define an adaptive system *Resilient Edge* to meet the application resiliency requirements using underlying LP-WAN technologies;
- provide open-source implementation of *Resilient Edge* and detailed insights considering hardware and network limitations.

The remainder of this paper is organised as follows: § II provides a technical background about the different LPWAN technologies. In § III, we define the adaptive *Resilient Edge* to meet the application resiliency requirements by providing two example applications. In § IV, we formulate a criticality-aware QoS allocation problem using Integer Linear Programming (ILP) and bin packing algorithms. § V provides the implementation details of *Resilient Edge* prototype and evaluate the baseline metrics. In § VI, we perform the evaluation of our prototype and discuss hardware and network limitations. In § VII and § VIII, we present related work and conclusion, respectively.

II. Background

In this section, we provide a background on multi-mode communication network technologies such as LPWAN technologies (LoRa, Sigfox, LTE (CAT-M1/NB-IoT)) and Wi-Fi that we use to provide resilience through redundancy in the *Resilient Edge* end-to-end system as shown in Figure 1. We provide a brief introduction to the technology, its range, use-case, security and energy-efficiency. We also provide various performance metrics (max payload length, the possibility of sending continuous data, latency, throughput, time to connect and reconnect) stated and observed in the wild in § V-A.

A. LPWAN Technologies

LoRa: LoRa is an Radio frequency (RF) modulation technology for low-power, wide area networks (LPWANs) protocol developed by Semtech. It has a range of up to 5 KM in urban areas and up to 15 KM or more in rural areas (line of sight) [9]. LoRa is suitable for specific use cases having requirements of long-range, low power, low cost, low bandwidth, secure with coverage everywhere. For example, measuring water flow using a water flow meter [10] sending data over LoRa.

A LoRa based network consists of end devices, gateways, a network server, and application servers. End devices send data to gateways (Up link (UL)) using single-hop LoRa or Frequency-shift keying (FSK) communication. The gateways send the data to the network server via a secured Internet Protocol (IP) connection, which, in turn, passes it on to the application server. Additionally, the network server can send messages (either for network management or on behalf of the application server) through the gateways to the end devices (Down link (DL)). LoRa allows intermediate gateways to relay messages between the end-devices to the network server, which routes it to the associated application server. Communication between the end-devices and gateway is performed on different frequencies and data rates, which is a trade-off between message length, communication range [11]. The data transfer from the end device to the application server is encrypted using Advanced Encryption Standard (AES) [12].

From the energy-efficiency perspective, LoRa devices have three classes [13]. Class A device can send data anytime and opens two receive windows after one and two seconds after an UL transmission. They are the most energy-efficient; however, the DL is only available after transmission. Class B is energy efficient with latency controlled DL. They utilize time-synchronized beacons transmitted by the gateway to sync up receive windows. Class C is not efficient in terms of power as they keep the receive window open after transmission [14]. LoRa also implements Adaptive Data Rate (ADR) by managing the data rate and RF output for each end-device individually to maximize battery life and maintain network capacity.

Sigfox: Sigfox uses publicly available and unlicensed bands to exchange radio messages over the air (868-869 MHz and 902-928 MHz). It uses Ultra-Narrow Band (UNB) technology combined with differential binary phase-shift keying (DBPSK) and Gausian FSK (GFSK) modulation. It has a range of approximately 10 km (urban), 40 km (rural). Sigfox mainly caters to IoT applications allowing small messages. For example, a letterbox sensor [15] sending a message to the user on receiving a post.

The end-device sends the message to the base stations (gateways), which forwards it to the Sigfox backend via a backhaul (3G/4G/digital subscriber line (DSL)/Satellite). The backend stores the messages to be retrieved by the end-user via browser/Representational state transfer (REST) Application Programming Interface (API) or set up a callback. For achieving high QoS, the end-device sends the message at a random frequency and then sends two replicas on different frequency and time (time and frequency diversity). The message can be received by any number of base stations (spatial diversity). However, Sigfox does not provide any authentication or encryption for the message and device [12].

From an energy-saving perspective, the end-device does not require pairing or sending synchronization messages to send the message, thus increasing battery life [16].

LTE (CAT-M1/NB-IoT): NB-IoT is a 3rd Generation Partnership Project (3GPP) radio technology standard designed for extended range operation, higher deployment density, and in-building penetration. It utilizes 180 kHz bandwidth and is deployed in-band, guard-band, or standalone mode. On the other hand, LTE-M provides high latency communication, support for extended coverage, LTE-M half-duplex mode/full-duplex, short message service (SMS), coverage enhancement, connected mode mobility [17]. Both NB-IoT and LTE-M have a range of approx. 1 km (urban) and 10 km (rural) [12]. Both follow 3GPP standards and have LTE encryption by default.

NB-IoT is suited for static, low throughput, and low power applications. For example, Nortrace tracked sheep's location and well-being over mountainous regions using NB-IoT [18]. In contrast, LTE-M is best for applications requiring mobility, voice, and SMS [19]. For example, Telstra tracks the location of high-value, non-powered assets, such as shipping containers, semi-trailers, rail freight wagons, and large machinery using LTE-M [20].

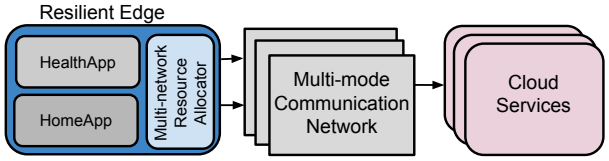

Fig. 1: Resilient Edge End-to-end System.

From the energy-efficiency perspective, both include Power Saving Mode (PSM) and Extended Discontinuous reception (eDRX). PSM reduces the energy used by User Equipment (UE) which defines how often and how long the UE will be active to send and receive data. eDRX improve end-device life for mobile-terminated traffic by switching off the receiver circuit for a defined period [21].

Integration - LoRa/Sigfox*:* NB-IoT/LTE-M is an IP-based network allowing data to be transferred to its associated cloud server. However, in the case of the LoRa and Sigfox, data is sent to The Things Network (TTN) console and Sigfox backend, respectively. Currently, TTN console and Sigfox backend provide multiple integration methods to retrieve data such as AWS IoT, AllThingsTalk, Microsoft Azure IoT Hub, HTTP, Emails, and other callbacks.

B. Wi-Fi

Mostly IoT devices have a low-cost, low-power system on a chip micro-controller with integrated Wi-Fi and dual-mode Bluetooth. Wi-Fi on IoT devices support different wireless modes such as 802.11 b/g/n/e/i, provide automatic beacon monitoring and scanning. The Pycom FiPy board used in our prototype has a Wi-Fi radio system on chip with $1KM$ Wi-Fi range.

III. System Model and Motivating Example

To better understand the network requirements of *Resilient Edge* applications, we start by considering a use-case with a concrete example. Figure 1 and 5 show an edge device running two sample applications to support assisted living facilities: one of the applications monitors the health of the resident (HealthApp), the other monitors their residential unit (HomeApp). In the real-world setting, the similar new applications can be configured for the data rates defined by the application QoS requirements and maximum network bandwidth availability. To achieve continuous network connectivity needed by these applications, we make use of a multi-mode communication network (details are provided in the next section).

We now present an abstract system model defining the attributes of application data flows so that we can reason about the QoS needs of each application, and about ways to (partially) fulfill those needs under different scenarios and different levels of multi-network connectivity. We propose that the communication needs of specific applications must be explicitly declared as message flows. An application can declare an arbitrary number of message flows, and each message flow represents a potentially infinite series of messages

TABLE I
Message flows on an edge device for assisted living facilities.
C: maximum message size (bytes)
T: minimum interval between subsequent message (seconds).

	Criticality Level	**1**		**2**		**3**	
Applications	**Message Flow τ**	**C**	**T**	**C**	**T**	**C**	**T**
HealthApp	1 fall detection	1000	10	40	20	10	60
”	2 heart monitoring	1000	5	80	10	10	20
”	3 body temperature	30	30	10	120		
HomeApp	4 sensor bedroom	40000	10	10	30		
”	5 sensor bathroom	80	10	10	30		
”	6 sensor lounge/kitchen	40000	10	10	30		
”	7 sensor front door	40000	10	10	30		
”	8 energy usage	40	3600				

to be sent through one of the local network interfaces. To allow application developers to quantitatively declare the QoS needs for each message flow, we revisit the notion of mixed-criticality communication proposed in [22] and support the definition of QoS requirements at distinct levels of criticality. As in [22], our goal is to allow the system to guarantee a predefined level of service for all message flows during normal operation, but also provide graceful degradation of service in adverse circumstances by allowing the most critical communication to be maintained. Unlike [22], however, we are not interested in meeting hard real-time deadlines and will instead use the notion of criticality-specific QoS requirements to manage multi-network resources.

Our model allows system designers and administrators to decide how many levels of criticality $L = L_{max}$ to support, and then to allow the specification of the QoS requirements of each message flow at each of those levels. The *Resilient Edge*, as shown in Figure 1 is designed to support three levels of criticality, and the Table I shows the QoS required by each message flow at each level. The message flows in Table I have been defined by taking a bottom to top approach. We assume that high criticality level messages are necessary to be delivered messages and are rare and have smaller size. In this example, for message flow 1, when criticality level 3 is requested and served, underlying network interface guarantees a message delivery service with message size of 10 bytes with 60 seconds subsequent message interval. The applications (designed by application developers) can request any message size and message interval; however, the values in our example reflect the prototype application data size and are intuitively set by authors considering several state-of-the-art IoT applications. Additionally, the message flows and the requirements are driven by the network capacity available on the edge device (e.g., FiPy[8] in our prototype).

We define $L = 1$ as the criticality level denoting normal operation mode, so the QoS requirements at that level should declare the largest communication volumes and injection rates of each message flow to account for all critical and non-critical traffic. QoS requirements at higher levels of criticality ($L = 2$ and $L = 3$) should only be declared for message flows that carry critical data and should account only for the necessary communication volumes and injection rates at each of those levels. By declaring or not a QoS requirement at a given level, application developers can explicitly distinguish the criticality of each message flow, and to explicitly define a number of

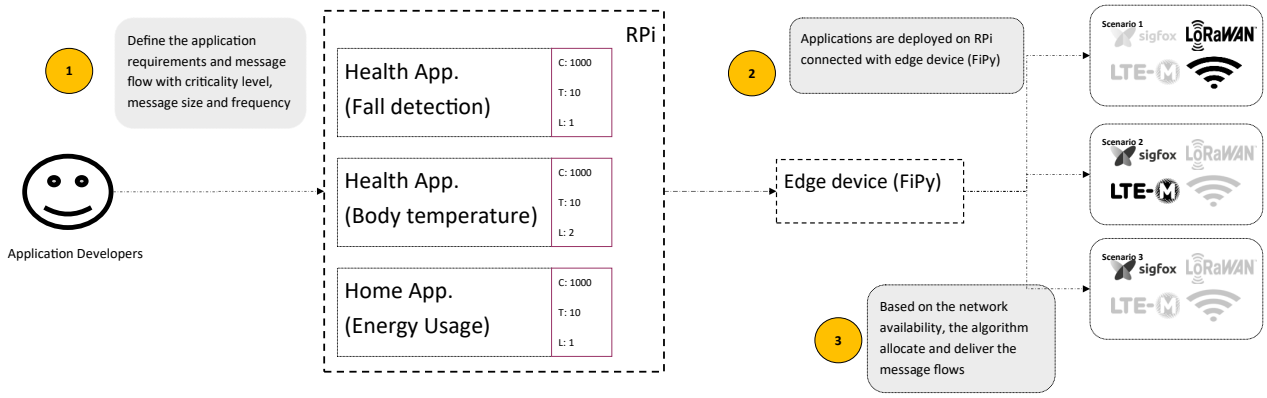

Fig. 2: System model summary with different applications with criticality, message size and frequency defined by application developers and different network availability scenarios (Faded symbol represents network unavailability).

service degradation levels each of them can support.

We can now define a message flow τ_i as a tuple (A_i, C_i, T_i) where A_i denotes the application to which the message flow belongs to, C_i denotes the maximum message size (in bytes) and T_i denotes the minimum interval between subsequent messages of the flow (in seconds). The bandwidth utilisation U_i of a flow τ_i can be calculated by the quotient C_i/T_i.

To support multiple criticality levels, C_i and T_i are defined as arrays of length L_{max}, so C_i^L and T_i^L denote, respectively, the maximum message size and the minimum interval between subsequent messages of τ_i at criticality level L.

In normal operation (i.e. $L = 1$), message flows declare their most generous QoS requirements, with larger data volumes for home monitoring (e.g. including camera snapshots in most of them) and resident monitoring (e.g. detail accelerometer data for fall detection, full electrocardiogram data for heartbeat monitoring). The next criticality level (i.e. $L = 2$) allows the declaration of degraded QoS levels, which in this example is provided for all message flows except for the one monitoring energy usage (which will not be forwarded by the edge device in case of degraded service). Notice that the QoS requirements declared for $L = 2$ show that monitoring will be performed less often and less data will be provided (e.g. simple movement detectors for home monitoring, average temperature and heartbeat for health monitoring). Finally, only two message flows declare QoS requirements at the highest level of criticality (i.e. $L = 3$), representing the alarms for fall or severe arrhythmia/cardiac arrest. In the case of degraded service, all available resources should be used to provide those two flows with their declared QoS requirements.

IV. Multi-Network Resource Management

Given the system model proposed in Section III, we can now formulate a criticality-aware QoS allocation problem.

A straightforward way to ensure QoS to the application message flows is to prevent the over-utilisation of the network interfaces they are assigned to. For example, by providing criticality level $L = 2$ guarantees to all message flows of the HealthApp application from Table I it would be possible to allocate all of them to a LoRa network (as their compound bandwidth utilisation would not exceed 6 bps), but the same network would be over-utilised if flows operate at criticality level $L = 1$ (where their compound bandwidth utilisation would exceed 1700 bps).

We can therefore formulate the criticality-aware QoS allocation problem as the choice, for each message flow of each application, of its allowed criticality level of service and its allocated network interface. Such problem is similar to a Variable Size Bin Packing Problem (VSBPP) [23], but with a fixed number of bins (i.e. the different networks, each of them with their bandwidth and payload size limitations) and with a choice of sizes for each element (i.e. the message flows, with their choice of criticality level).

A. ILP Formulation

Similarly to the standard VSBPP, we can formulate our problem with an ILP model. For the sake of simplicity, we describe the size of bins and elements by their bandwidth capacity and utilisation, respectively. We claim that an extension to a multi-dimensional formulation (i.e. that can also capture maximum payload sizes, maximum number of daily messages, etc.) is straightforward but left as future work. The assumption is that the QoS requirements of all applications can be satisfied provided there is enough bandwidth of one network or combined bandwidth of multiple networks. However, in practice, the network capacity is limited, and there would be applications whose QoS requirements cannot be satisfied.

Let us then consider a set $\mathcal{T}$ of elements representing our message flows $\tau_i, i = 1...n$, each of them with a potential choice of values U_i^L representing the different bandwidth utilisations C_i^L/T_i^L at each level of criticality they are designed to support (or ∞ if that flow does not specify service at a given criticality level, e.g. *energy usage* flow at levels $L = 2$ and $L = 3$ in Table I).

Likewise, let us consider a set Γ of bins representing our network interfaces $\gamma_j, j = 1...m$, each of them with a bandwidth capacity B_j. Finally, we define a set of binary variables $x_{i,j,L} \in \{0, 1\}$, and assume that $x_{i,j,L} = 1$ if message flow τ_i is assigned to network γ_j and configured to operate at criticality level L, or $x_{i,j,L} = 0$ otherwise. Given

the ranges $1 \leq i \leq n,\ 1 \leq j \leq m,\ 1 \leq L \leq L_{max}$ we will have at most $n \times m \times L_{max}$ binary variables for a given problem.

To ensure the assignment of values to the binary variables represent a valid solution to our problem, we must now state a number of constraints. First, we make sure that a message flow τ_i is allocated to a single network interface and configured to operate at a single criticality level by stating that $\sum_{j=1}^{m} \sum_{L=1}^{L_{max}} x_{i,j,L} = 1$ for all $1 \leq i \leq n$. Secondly, we ensure that no network interface γ_j is overloaded by stating that $\sum_{i=1}^{n} \sum_{L=1}^{L_{max}} x_{i,j,L} \times U_i^L \leq B_j$ for all $1 \leq j \leq m$.

Finally, we can state our maximisation objective function as:

$$objective = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{L=1}^{L_{max}} x_{i,j,L} \times (1 + L_{max} - L) \qquad (1)$$

The rationale behind the maximisation of the objective is to configure message flows at the lowest possible levels of criticality (i.e. lowest values for L), thus providing each message flow with the most generous possible QoS, while avoiding network overload. The unit added to the last term of the equation is crucial to allow the objective to distinguish a flow that is allocated at the highest criticality and one that is not allocated at all.

An additional constraint could be formulated, in case all message flows must be allocated to a network interface and receive some level of service: $\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{L=1}^{L_{max}} x_{i,j,L} = n$. This is not always necessary or desirable, as it may the intention of application designers that, under limited network availability, only a subset of the application message flows should be provided service (e.g. in the example from Table I, under the most stringent conditions at $L = 3$, only the *fall detection* and *heart monitoring* message flows require service). In such cases, such a constraint may be rewritten to ensure that specific message flows are always allocated service, or even be reformulated as part of the objective function, aiming to maximise the number of message flows that are guaranteed some level of service.

B. Bin-Packing Algorithms

While the formulation given in subsection IV-A can be optimally solved by an ILP solver, it may not be reasonable to expect that such a software package could be installed and executed by resource-constrained edge devices such as the ones considered in this work. We, therefore, propose the use of simple bin-packing algorithms that are able to achieve acceptable results with a much lower computational overhead. In particular, we define a criticality-aware best fit (CABF) algorithm and show its performance compared to classic first fit, best fit, and worst fit algorithms (FF, BF, and WF) as well as their decreasing variants (FFD, BFD, and WFD).

Since the classic algorithms are unaware of the different criticality levels, we implemented two alternatives for each of them, one that tries to fit message flows to networks at their highest level of criticality (i.e. H-FF, H-BF, H-WF and their decreasing counterparts) and another that does the same with the lowest defined criticality of each message flow (i.e. L-FF, L-BF, L-WF and their decreasing counterparts).

Algorithm 1: Criticality-Aware Best Fit (CABF)

Result: Set of 3-tuples indicating the allocated network and configured criticality level for all message flows that can be provided service

CABF $(\mathcal{T}, \Gamma)$
- **inputs:** set $\mathcal{T}$ of message flows, set Γ of networks
- **output:** set Q of 3-tuples $q = (\tau_i, \gamma_j, L)$
- $Q \leftarrow \emptyset$;
- **for** $(l = L_{max};\ l > 0;\ l = l - 1)$ **do**
  - **foreach** $(q \in Q \mid q(\tau) \in \mathcal{T}_l \wedge q(L) > l)$ **do**
    - $Q \leftarrow Q - q$;
    - $\gamma_{reloc} \leftarrow BestFit(q(\tau), l, \Gamma)$;
    - **if** $\gamma_{reloc} \neq \emptyset$ **then**
      - $q \leftarrow (\tau, \gamma_{reloc}, l)$;
    - $Q \leftarrow Q + q$;
  - **foreach** $(\tau_{new} \in \mathcal{T}_l \mid \tau_{new} \notin Q(\tau))$ **do**
    - $\gamma_{new} \leftarrow BestFit(\tau_{new}, l, \Gamma)$;
    - **if** $\gamma_{new} \neq \emptyset$ **then**
      - $Q \leftarrow Q + (\tau_{new}, \gamma_{new}, l)$;
- **return** Q;

Algorithm 1 describes the proposed CABF algorithm, which takes as inputs the sets $\mathcal{T}$ of message flows and Γ of networks, and outputs a set Q of 3-tuples $q = (\tau_i, \gamma_j, L)$, each of them representing the allocation of a message flow τ_i to a network γ_j at criticality level L. Algorithm 1 uses the following notation: $q(\tau)$, $q(\gamma)$ and $q(L)$ denote the first, second and third element of a 3-tuple q, and likewise $Q(\tau)$, $Q(\gamma)$ and $Q(L)$ denote the sets of all first, second and third element of the 3-tuples in Q; $\mathcal{T}_L$ is the subset of $\mathcal{T}$ including all message flows that are declared at a given criticality level L (as not all flows must be declared for all levels); and $BestFit(\tau, L, \Gamma)$ denotes a function which returns the network $\gamma \in \Gamma$ which is the best fit allocation for the message flow τ at its criticality level L, or $\emptyset$ if τ does not fit in any of the networks in Γ, taking into account the allocations already in Q.

The intuition behind the CABF algorithm is as follows. It tries to allocate first the message flows defined at higher criticalities, as shown by the outer *for* loop decreasing from L_{max} to 1. As it iterates over that loop towards lower criticality levels, and before it allocates flows defined at the criticality level of the current iteration, it first attempts to lower the criticalities of flows allocated in the previous iterations. This is shown by the first inner *forall* loop, which iterates over tuples in Q with flows that have definitions at the criticality level of the current iteration (i.e. $q(\tau) \in \mathcal{T}_l$). Within the first inner *forall* loop, the algorithm removes the original allocation from Q, then tries to find a network γ_{reloc} which is the best fit for the flow using its lower criticality figures. If the best-fit algorithm succeeds to find an allocation with

TABLE II
Obtained criticality level (1 | 2 | 3) and network allocation (* Wi-Fi | # Lora | + Sigfox) for motivating example

	Message Flows								% flows served	avg crit level	objective
	1	2	3	4	5	6	7	8			
Requested crit level	1,2,3	1,2,3	1,2	1,2	1,2	1,2	1,2	1			
Allocation algorithms	**Allocated Criticality Level**										
L-FF	1*	1*	1*	1*	1*			1*	75	1	18
L-FFD		1#	1#	1*	1#	1*		1+	75	1	18
H-FF	3*	3*	2*	2*	2*	2*	2*	1*	100	2.12	15
H-FFD	3*	3*	2*	2*	2*	2*	2*	1*	100	2.12	15
L-WF	1*	1*	1*	1*	1*			1*	75	1	18
L-WFD		1#	1#	1*	1#	1*		1+	75	1	18
H-WF	3*	3*	2*	2*	2*	2*	2*	1*	100	2.12	15
H-WFD	3*	3*	2*	2*	2*	2*	2*	1*	100	2.12	15
L-BF	1#	1*	1+	1*	1#			1+	75	1	18
L-BFD		1#	1+	1*	1#	1*		1+	75	1	18
H-BF	3+	3+	2+	2+	2+	2+	2+	1+	100	2.12	15
H-BFD	3+	3+	2+	2+	2+	2+	2+	1+	100	2.12	15
CABF	2+	1#	1+	2+	1#	1*	1*	1+	100	1.25	22
CABFinv	1#	2#	1+	2+	1#	1*	1*	1+	100	1.25	22
Optimal	1*	1*	1*	1*	1*	2*	2*	1*	100	1.25	22

the lower criticality values, a 3-tuple representing that new allocation is added to Q. If it fails to find a network that is able to accommodate the requirements at a lower criticality, the original allocation is returned back to Q. Once the first inner *forall* loop finishes, the second inner *forall* loop uses the best-fit algorithm to allocate, when possible, all unallocated message flows that have definitions at the criticality level of the current iteration.

The proposed order of the two inner *forall* loops reflects an assumption that flows that have definitions at higher levels of criticality should always be given more resources if they become available. This will not always be the case in every application domain, and in many cases it may be better to first use resources to provide some service to less-critical message flows rather than improve the service to highly-critical ones. Reversing the proposed order of the two inner *forall* loops would achieve exactly that, therefore we name that variant $CABF_{inv}$.

C. Evaluation - Motivating Example

Table II shows the network allocations and choice of criticality level for each of the message flows of the motivating example described in Section III. The table shows allocations produced by each of the baseline bin-packing algorithms, by both variants of the proposed algorithm, and one solution (out of many possible ones) produced by an optimal solver. The allocations assume the availability of three networks with bandwidths of 64000, 1760 and 48 bits per second, representing Wi-Fi, LoRa SF9 and Sigfox networks (but disregarding maximum payload size or the number of daily messages), and represented by the symbols $*$, $\#$ and $+$, respectively.

Both variations of the proposed algorithm are able to produce optimal solutions in this example, providing service to all flows, with all-but-two at their lowest criticality level (which leads to an objective result of 22 according to Equation 1).

D. Evaluation - Synthetic Applications

To show the superiority of proposed algorithms over a much larger number of examples, we created hundreds of synthetic application models and compared the performance of the proposed algorithms against all the baseline bin-packing algorithms described in subsection IV-B. Each synthetic application has a well-defined number of message flows *nFlows*, and a number of available criticality levels *nCrit*. Just as in the motivating example, message flows do not have their QoS requirements defined at every level of criticality (as they may be completely dropped in case of severe network degradation). To model that, the generation of synthetic applications uses a probability factor $0 < hC < 1$ that determines if a given message flow has a definition for any given criticality level above normal operation (i.e. $L > 1$). The factor is re-applied for each additional level of criticality, so for example a factor *hC* = 0.8 means that a message flow has an 80% chance of having a QoS definition for L=2, 64% chance for L=3, 51.2% for L=4, and so on. A multiplicative factor $0 < multC < 1$ is then used to generate the QoS requirements of the message flow at a higher criticality level (by multiplying the period *T* and maximum packet size *C* requirements defined at the preceding level of criticality).

To perform our experiments, we generated four sets of 100 synthetic applications. All applications within the first set have 10 message flows each (*nFlows* = 10) and are therefore similar to our motivating example, which has 8 flows. Applications in the other three sets are much larger, with *nFlows* = 20, 40 and 80, respectively. The QoS requirements under normal operation were generated for each message by uniformly sampling a range of periods (5 to 120 seconds) and maximum message sizes (7 to 260 kilobytes). The number of available criticality levels *nCrit* was set to 4, the value for the *hC* factor was set to 0.8, and the value of the factor *multC* was uniformly sampled for each flow from an interval between 0.4 and 0.8.

Again, we assumed a platform with three networks with available bandwidths of 64000, 1760 and 48 bits per second.

Once we generated all applications and their respective message flows, we used each bin-packing algorithms described in subsection IV-B to decide which network interface should be allocated to each message flow, and which level of criticality should be supported. For each allocation of each application, we recorded two metrics:

- *servP* - the percentage of traffic flows of the application that were provided some level of network service.
- *avCL* - the average criticality level that was supported for the flows of the application.

The *servP* metric is straightforward: for an application with *nFlows*=20, a *servP* value of 60% denotes that 12 of its flows were allocated to a network interface. The *avCL* metric is slightly more complex, as it has to be expressed as a percentage of the highest criticality level supported by the application (since not every application has flows with QoS requirements defined at all possible levels of criticality). This means that, for example, if an application has its flows defined at 4 criticality levels, and the average criticality level assigned to all its flows is 1.4 (i.e. the sum of the criticality level assigned to each flow, divided by *nFlows*), the value of *avCL* would be 35% (i.e. 1.4 is 35% of 4). If that same application had the same average criticality of 1.4, but its flows only had

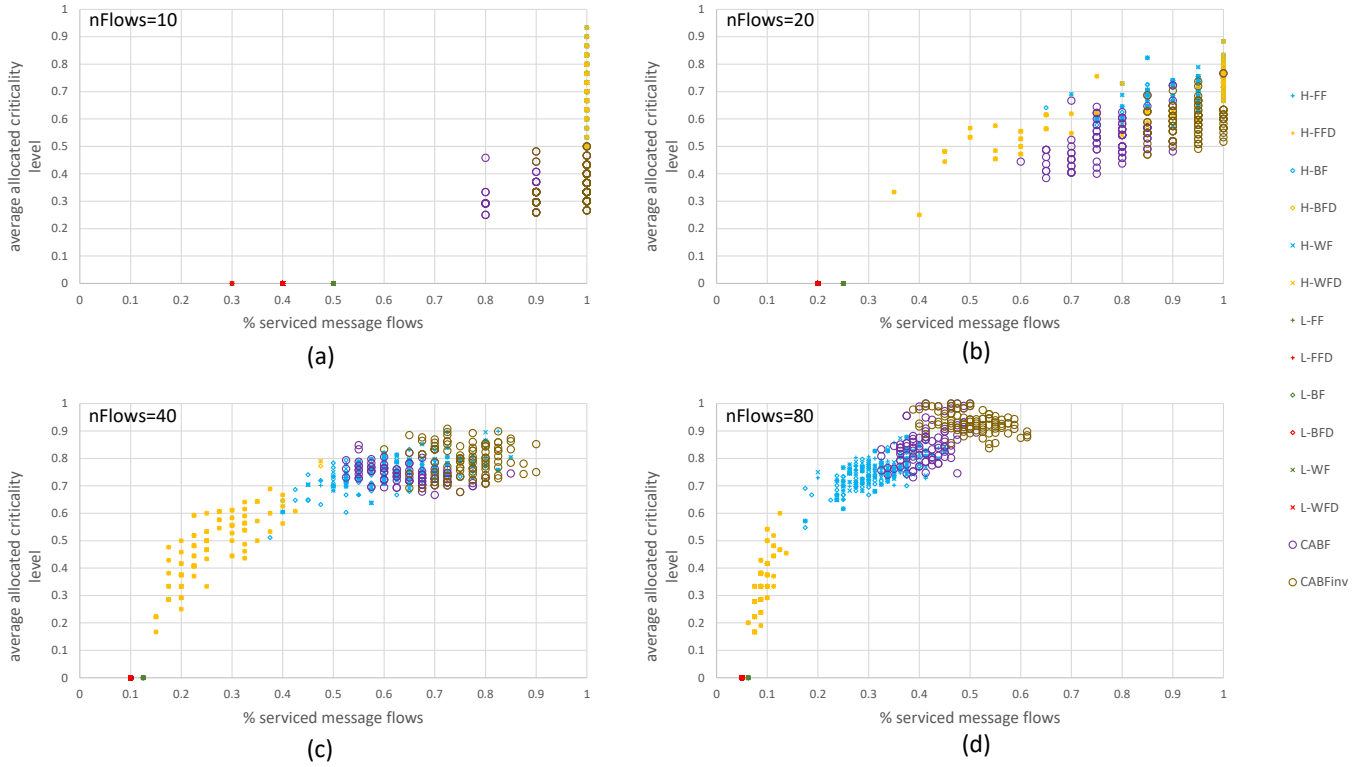

Fig. 3: Experimental results for synthetically generated applications with (a) 10, (b) 20, (c) 40 and (d) 80 message flows each.

definitions up to 3 criticality levels, the value of *avCL* would then be 46.66

Figure 3 shows four plots, one for each of the sets of 100 applications. The number of flows per application is shown in the upper-left corner of each plot. The metric *servP* is plotted against the x axis, and the metric *avCL* against the Y axis. Every point in the plot represents one allocation of an application of the set, and the shape of the point shows which bin-packing algorithm was used for that particular allocation. That means that each plot should have 1400 points (100 applications allocated using 14 different bin-packing algorithms), but in most cases that number of visible points is much smaller because multiple allocations and multiple applications actually have the same values for both metrics and therefore overwrite one another on the plot.

In the optimal case, a bin-packing algorithm would produce the highest possible value for the *servP* metric (meaning all flows were provided some service) and the lowest possible value for the *avCL* metric (meaning that flows were allocated QoS service levels that were the closest possible to *L=1*, i.e. normal operation). Given the large number of applications and message flows considered in this evaluation, it was not feasible to solve each case to optimality, so all the results achieved by the proposed algorithms and baselines are a trade-off between both metrics.

Therefore, the key findings of this experiment are obtained by observing which bin-packing algorithms produced the best allocations: those at the lower-right corner of the plot, with the highest values for *servP* and the lowest values for *avCL*.

In the plot for the set with *nFlows*=10, we can see that many algorithms were able to provide network service to 100% of the flows of most applications, albeit at a high level of criticality in the case of many of the *H* bin-packing variants (i.e.blue and yellow markers). As expected, the *L* bin-packing variants were able to allocate message flows at their lowest criticality level (i.e. normal mode), but could only allocate a small percentage of them before reaching the saturation of all available network interfaces.

As we increase the number of message flows per application (i.e. plots with *nFlows*=20 and 40), we can see that the additional workload to be allocated pushes the results away from the lower-right part of the plot: allocations either have lower *servP* (i.e. fewer message flows are allowed access to a network interface) or higher *avCL* (i.e. message flows are allocated at higher levels of criticality, anf therefore more restrictive QoS levels). It is clearly noticeable, however, that the two proposed algorithms CABF and CABFinv are consistently producing the results that are closest to the optimal lower-right corner.

With the largest applications (*nFlows*=80), we can see that the proposed algorithms are the only ones that are able to provide network service to more than half of the flows, for some applications. In this scenario, it is also possible to see clearly the distinct behaviours of CABF and CABFinv: the former tries to allocate flows with more generous QoS levels (i.e. lower criticality), while the latter puts emphasis on providing service to as many flows.

V. Resilient Multi-network Edge Platform

This section describes a resilient multi-network edge platform we have designed and implemented, aiming to validate the concepts proposed in the previous sections over real off-the-shelf hardware and using realistic network deployments. Firstly, we describe the chosen hardware platform based on a Raspberry Pi and a Pycom FyPy communication board. The FyPy board supports multi-network connectivity over five different networks Wi-Fi, Bluetooth, LoRa, Sigfox and LTE (CAT-M1/NB-IoT). The Raspberry Pi, in turn, handles the proposed multi-network resource management approaches. A detailed description of the functionality of each board, and their integration will be provided, as well as their interoperation with the cloud to perform realistic data transfer scenarios. Additionally, we will provide details about key performance metrics that show the strengths and weaknesses of each type of network supported by the platform, namely maximum payload length, inter-message gap, latency, throughput, connection and reconnection time. The detailed description of our multi-network edge platform is then followed by a practical evaluation, where we implement the proposed multi-network resource management algorithms from Section IV.

Platform Overview

The *Resilient Edge* prototype setup is shown in Figures 4 and 5. A Raspberry Pi model 4 [24] (RPi) is interfaced with Pycom FiPy [8]. RPi has Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz with 4 GB RAM, FiPy has an Xtensa® dual-core 32–bit LX6 microprocessor and on-chip SRAM of 520KB and external SRAM 4MB with an external flash of 8 MB. FiPy provides connectivity to five different networks Wi-Fi, Bluetooth, LoRa, Sigfox and LTE (CAT-M1/NB-IoT). More details about the interworking of FiPy can be found in the FiPy datasheet [25]. The RPi Universal Asynchronous Receiver/Transmitter (UART) (GPIO14-TXD/ GPIO15-RXD) is connected to the expansion board pins (P3-TXD/P4-RXD) of FiPy to transfer the data from the RPi to FiPy. We implemented and simulated the message flow of HealthApp and HomeApp on RPi and multi-network resource allocator on FiPy, respectively. To enable the data transfer between RPi and FiPy, a message payload from the applications is written to the RPi UART and read by FiPy continuously. On FiPy, a python script checks the messages received from the RPi, the network interface assigned to the message flow, its criticality level, and attempt to send it via that network interface.

Implementation details of RPi components: We utilise Transmission Control Protocol (TCP)/IP serial bridge[1] to create a socket listening on port 8080 connecting to the UART (/dev/ttyAMA0) to send and receive data to and from the UART. Further, we utilise *python select lib* [2] to monitor sockets for incoming data to be read and send outgoing data when there is room in the buffer and utilise message queues to store the outgoing messages. To send and receive a message over UART efficiently and without breaking, we add a header with `(:ML:<MessageLength>)` at a start and a newline `'\n'` at the end. On both sides, RPi reading a socket and FiPy reading the UART, we ensure that we have received the full message. To simulate the message flows running on the RPi, we utilise threads to write a message payload on the socket with `<MessageFlow Name, Criticality level, the payload>`. The thread sleeps for the period specified by the message flow before sending the next message. We store the statistics about the number of messages sent by a particular message flow, acknowledgment or error received.

The FiPy runs a multi-network resource allocator and sends an allocation message back to the RPi stating which message flows have been assigned with which criticality level. A sample Message Flow Element Allocation (MFEA) message is:

```
MFEA:['PS': 41, 'N': 'Wi-Fi', 'PE': 10,
'MF': 'Kitchen Sensor', 'CL': 1]
```

where PS is payload size, N is network assigned, PE is the period (time in seconds) between two subsequent messages, MF is message flow name, and CL is criticality level assigned. If the network conditions at the FiPy are changed, and a new MFEA message is received, the previous thread sending the messages are stopped, and new threads for sending a message with a particular criticality level and period are launched.

Implementation details of FiPy components: On FiPy, before assigning any network to a message flow, we need to create a network `"bin"` of the available networks (Wi-Fi, LTE (CAT-M1/NB-IoT), LoRa and Sigfox) and add the corresponding network interfaces to the network `"bins"`. While doing so, we take into some limitations that are posed by underlying hardware such as if Wi-Fi is available, we do not add a network `"bin"` for LTE (CAT-M1/NB-IoT) because in the current version of FiPy if both Wi-Fi and LTE (CAT-M1/NB-IoT) are connected at the same time, FiPy does not provide routing capabilities to direct the traffic [26]. If Wi-Fi is unavailable, then we connect via LTE (CAT-M1/NB-IoT). Similarly, if the LoRa network is available, we add LoRa to the network `"bin"`. If LoRa is unavailable, we add Sigfox, mainly because Sigfox and LoRa share the same radio module. As part of the Multi-network resource allocator - we implement variant Criticality-Aware Best Fit ($CABF_{inv}$) and set the initial parameters, and perform the allocations of the message flows to the network interface. After the allocations, we continuously read the UART for the messages from the RPi. The messages from the RPi are in the format of `<MessageFlow Name, Criticality Level, Payload>`. On the FiPy, we check if the Message Flow with criticality level has been assigned; if assigned, an attempt to send the payload is made. If the message flow is not allocated, an error message is sent back to RPi, mentioning message flow is not allocated. Similarly, if the payload is delivered, an ACK message is sent to the RPi; if not delivered, an error message with not delivered is sent through UART.

[1]TCP/IP - serial bridge https://pyserial.readthedocs.io/en/latest/examples.html#tcp-ip-serial-bridge

[2]Select - Waiting for I/O completion https://docs.python.org/3/library/select.html

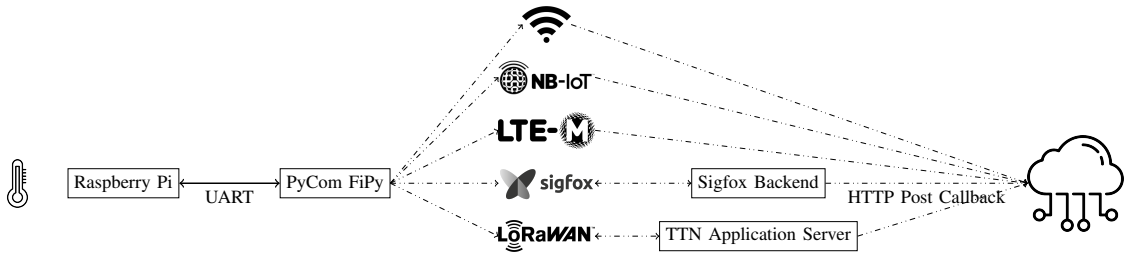

Fig. 4: Block diagram of current experimental setup.

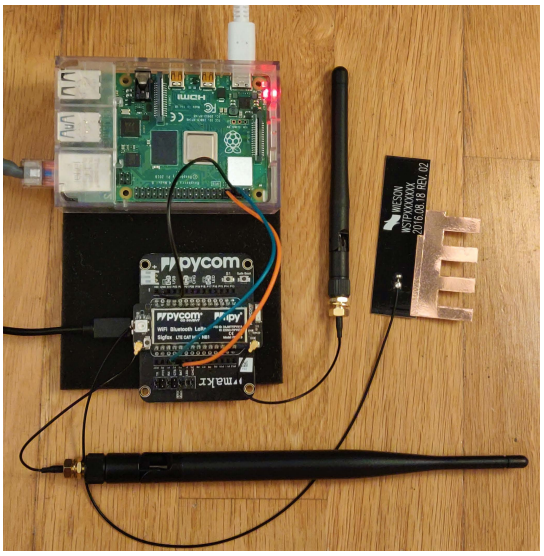

Fig. 5: Current Experimental Setup.

FiPy provides multi-network connectivity, and powering on all the network interfaces could result in significant power consumption. With that in mind, currently, we initialize all the network interfaces at the boot and connect to a specific network based on network availability and conditions. For instance, the NB-IoT connection is skipped if the Wi-Fi network is available; if the LoRa network joins successfully, the Sigfox socket is not created. Further, we have mentioned the time (in seconds) for different technologies to connect to the network (§ V-A) and time complexity and context switching of the $CABF_{inv}$ algorithm (§ VI-B) that provides a rough estimate on switching overhead if the IoT devices need to switch between network interfaces and to turn on/off the interface.

The multi-network resource meets the QoS requirements of IoT applications by determining the different network interfaces available and the communication parameters of the selected technology (bandwidth). For instance, when a network interface is defined (whether it is available or not), we determine the bandwidth provided by that network. For instance, LoRa starts the connection with adaptive Spreading Factor (SF), i.e., it would start with SF7; if it did not connect with SF7, it would try with SF8 and so on. Based on the connection, we take the bandwidth of the connected SF.

Receiving messages on Cloud: To store the messages sent by the FiPy (as shown in Fig. 4), we utilise Tornado - a python web framework and asynchronous networking library [3] to run an `HTTP` server on a machine hosted on a cloud. The `HTTP` server accepts `HTTP POST` messages and receives them directly from the FiPy via Wi-Fi, TTN application server via LoRa, Sigfox backend via Sigfox and Pybytes[4] via NB-IoT. When the message flow allocated interface is Wi-Fi, an `HTTP POST` request is sent from FiPy to the cloud machine using `urequests micro-python library`. When the assigned interface for message flow is LoRa, Sigfox and NB-IoT, the message is sent via the respective interface. On TTN application server, Sigfox backend and Pybytes for NB-IoT, we have configured the `HTTP` Integration as defined in § II-A. HTTP integration sends the UL data received from FiPy to our cloud machine. The `HTTP` server checks for the `URI` and fetch the data from the post data and stores it in a `influxdb` database.

A. Platform Metrics

For performance evaluation, we considered the following metrics: maximum payload length, inter-message gap, latency, throughput, time to connect and reconnect, and performed the initial experiments to get the baseline results for each network (LoRa, Sigfox, NB-IoT, Wi-Fi) before deploying the multi-network resource allocator on FiPy. In Table III, we provide a summary of the metrics found in these baseline experiments. Application developers can decide on the suitable network medium for the application based on the application requirements and the use-case. First, we provide how each network compares with each other, followed by more information about the experiment.

Maximum Payload Length: Maximum payload size determines how much information (in bytes) can be sent in one message and helps to determine the suitability for an IoT application. For *LoRa*, the max payload size varies from 51 bytes to 222[5]/242 bytes based on the configuration settings. On the other hand, *Sigfox* allows an UL payload of up to 12 bytes and a limit of up to 140 messages per day bytes payload with a limit of 4 messages per day DL. Most suitable from the payload perspective, is *NB-IoT/Wi-Fi*. LTE Transport block sizes (TBS) can support a maximum of 85 bytes DL and 125 bytes UL. However, as TCP/UDP protocol is used in Wi-Fi/NB-IoT, the payload is sent as multiple packets (the

[3]Tornado Web Server https://www.tornadoweb.org/en/stable/

[4]Pybytes https://docs.pycom.io/pybytes/

[5]The payload size is 222 bytes when the device is a repeater and requires optional FOpt control field [27].

TABLE III
Baseline metrics summary

Metrics	LoRaWAN	SigFox	LTE (CAT-M1/NB-IoT)	Wi-Fi
Max-payload length	1 - 222 bytes	1-12 bytes	UDP/TCP/IP	UDP/TCP/IP
Sending continuous data	0.165 ms	10.5 s	1-100 ms	1-100 ms
Latency	24 - 2800 ms + 1-100 secs	1 - 4.5 s	500 ms (avg)	8 ms (avg)
Throughput	250 - 11000 bps	UL: 100 bps DL: 600 bps	NB-IoT: UL: 66 kbps; DL: 26 kbps LTE-M: DL: 300 kbps; UL: 380 kbps	Local: 3550 Kbps Remote: 770 Kbps
Time to connect to network	OTAA: 5.6 s ABP (join not required)	1-100 ms	With LTE Reset: 20 s Without LTE Reset: 15.5 s	7.7 s

TABLE IV
LoraWAN airtime for max payload in Europe [28]

Configuration	Bitrate (bits/sec)	Max payload size (bytes)	Time on Air (ms)	Max number of messages/day
SF12/125	250	51	2793.5	12
SF11/125 kHz	440	51	1560.6	23
SF10/125 kHz	980	51	698.4	51
SF9/125 kHz	1760	115	676.9	53
SF8/125 kHz	3125	222	655.9	54
SF7/125 kHz	5470	222	368.9	97
SF7/250 kHz	11000	222	184.4	195

TABLE V
Sigfox Payload time approximate time provided by Sigfox [33] and observed for Average, Good/Excellent Quality at RC1 Region

	Stated	Observed	Observed
Payload Length	Approximate (sec)	Average (sec)	Good/Excellent (sec)
<1 bit	1.1	2	1
2 bit - 1 byte	1.2	1.6	2.0
2-4 byte	1.45	2.3	2.1
5-8 byte	1.75	4.5	2.5
9-12 byte	2	4.5	3

size and number of which depend on the path Maximum Transmission Unit (MTU)). So, the payload length for NB-IoT/Wi-Fi is bounded by the memory assignment capability of the device.

Table IV represents the max payload sizes with max number of messages per day at different SF/bandwidth and respective airtime for LoRa [29]. We use TTN, a public community network having a fair access policy [30] that limits the UL airtime to 30 seconds per day per node and the DL messages to 10 messages per day per node. The max number of messages in Table IV is calculated based on the 1 percent duty and the fair usage policy with maximum payload message. Further, to utilize application payloads efficiently, LoRa best practices [31] to limit application payloads can be referred.

Sigfox provides Link Quality Indicator (LQI) [32] based on the Received Signal Strength Indicator (RSSI) and number of base stations that received a message. However, as only four DL messages per day are allowed, it is advisable to set up an HTTP/Email callback to get service-related information.

***Inter-message gap**:* We conducted this primitive experiment to understand the limitation of the time between sending two consecutive messages. For *LoRa*, on average it took $0.165\ ms$ to send a message. For *Sigfox*, in terms of sending a continuous message on Pycom FiPy end-device, it takes around an average of $10.5\ s$, with the minimum $9\ s$ and maximum $12\ s$ to send a message on Sigfox in RC1 region with $100\ bps$. Suppose the application priority is to send the messages fast. In that case, sending a message via Wi-Fi and NB-IoT takes a few milliseconds.

To experiment, for LoRa, we sent 40 messages with different payloads, ten messages with four SF options offered by LoRa, i.e., (SF7 - 1 byte, SF12 - 1 byte, SF7 - 242 bytes, SF12 - 51 byte). For Sigfox, a message with a 12-bytes payload takes $2.08\ s$ over the air with a rate of $100\ bps$. Further, the device emits a message on a random frequency and then sends two replicas on different frequencies and time [16]. We experimented with sending continuous data on Sigfox of variable length starting from 1 byte to 12 bytes. We measured the time before sending the message using `'socket.send(msg)'` and after that. We sent 60 (5×12) messages, three times on average LQI, and one time each on good and excellent LQI.

***Latency**:* We define latency as the delay between transmitting a packet and its arrival at its destination. It combines transmission, propagation, and processing time at both ends. For *LoRa*, TTN latency ranges between $24\ ms$ (smallest payload - fastest bit-rate) to $2800\ ms$ (max-payload on slowest bit-rate) from the end-device to the gateway. For Sigfox, Table V provides the approximate time taken by the message to reach Sigfox backend from the edge device provided by Sigfox [33] and observed time taken by payload of different sizes at different LQI (average/good/excellent) in RC1 region for Sigfox. For NB-IoT, on average, it has a latency of $576\ ms$. It is important to mention that when ping is used the first time, the latency is high in the range of $10\ s$ and then stabilises slowly (after $5 - 10$ pings) to the range of $500 - 800\ ms$. From literature, NB-IoT latency ranges around $1 - 10\ s$ [34] depending on normal coverage or extended coverage. Latency in LTE-M is around $100 - 150\ ms$ in normal coverage. For *Wi-Fi*, latency has an average of $8.32\ ms$ and $16.70\ ms$ with a standard deviation of $9.93\ ms$ and $12.19\ ms$ for the machine in local and remote networks, respectively. Fig. 6 provides latency of the Wi-Fi network when FiPy pings a machine in the same local network and remotely in the cloud network. The network latency of NB-IoT varies significantly compared to the Wi-Fi.

For *LoRa*, the transit time from the gateway to the application completely depends on the solution implemented. On TTN and with a gateway connected through wired Ethernet,

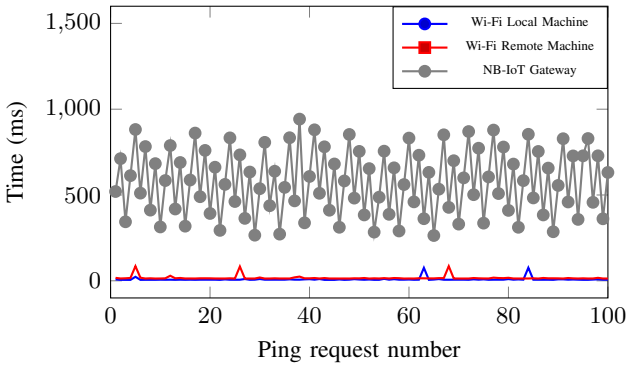

Fig. 6: Latency results for pinging a local machine and cloud machine via Wi-Fi and gateway via NB-IoT network.

TABLE VI
Sigfox radio configuration [39]

Frequency (MHz)	**RC1/RC3/RC5**	**RC2/RC4**
Uplink center	868.130/923.200/923.300	902.200/920.800
Downlink center	869.525/922.200/922.300	905.200/922.300
Uplink data rate (bit/s)	100	600
Downlink data rate (bit/s)	600	600

it will take tens of milliseconds (at the current load levels). If the gateway uses a slow cellular connection, the delay will increase. Further, up to a few seconds can add up based on the selected callback mechanism (HTTP, AWS IoT, others). At a high level, latency would be a sum of time-on-air, gateway to network server network latency, duplication window, routing services processing time, a selected callback to application network latency. LoRa TTN fair usage policy only allows at most 10 DL messages. If we also consider the DL latency, one or two seconds could be added to the latency as there are two receive windows after a UL message. For *Sigfox*, to understand the latency, we calculated the time when we started sending the message using the device and when it was received at the Sigfox backend. We synced the end-device time using Network Time Protocol (NTP) with `pool.ntp.org` server. For *NB-IoT*, we connected to the NB-IoT Vodafone network with Pycom provided subscriber identity module (SIM) [35] having pycom.io Access Point Name (APN). We figured out our IP Address using AT command 'AT+CGCONTRDP' and sent around 100 ping requests to the gateway, which was three hops away (calculated from TTL). For Wi-Fi, we connected the end-device FiPy to the home Wi-Fi network and calculated the latency by sending 100 $uping$ [36], [37] requests to a local machine in the same network and to a remote machine on a cloud.

Throughput: This experiment measured the average throughput (bits per second) achieved on each network individually. For *LoRa*, bit-rate depends on the bandwidth and SF. In Europe, the regional parameters [38] allow a bandwidth of 125 KHz to 250 KHz and SF of $7-12$ [27]. LoRa data rates range from 0.3 $Kbps$ to 50 $Kbps$ [11]. For *Sigfox*, Table VI provides UL and DL frequency and data rate for different regions. Based on the Sigfox frequency, the data rate could be determined. For *NB-IoT*, data rate [34] is 26 $Kbps$ in the DL, and 66 $Kbps$ in the UL. LTE-M has approximately

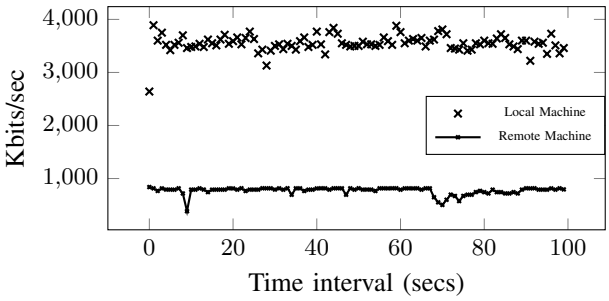

Fig. 7: Bandwidth results using iperf when running on local network and cloud.

300 $Kbps$ in DL and 380 $Kbps$ in the UL in half-duplex. On an average on the field, 100 to 150 $Kbps$ are reached in both directions. For Wi-Fi, bandwidth has an average of 3550.8 $Kbps$ and 770.181 $Kbps$ with a standard deviation of 157.19 $Kbps$ and 71.86 $Kbps$ when $iperf3$ is hosted locally in the local network and cloud network, respectively. Fig. 7 provides the bandwidth of the Wi-Fi network when FiPy pings and connects to the `iperf` server in the same local network and remotely in the cloud network.

For our experiments, for NB-IoT, we are using Pycom provided Vodafone SIM; the User Equipment (UE) can only communicate to a white-listed IP address because of which we were unable to host an instance of `iperf` on a server and calculate throughput. For *Wi-Fi*, Pycom FiPy utilises `ESP32` which provides 20 $Mbps$ TCP RX/TX in the test [40] performed in the lab. The bandwidth and throughput was calculated using `uiperf3` [41].

Time to connect to the network: We conducted baseline experiments to understand the connection time an end device takes to join the different networks. It helps to estimate switching overhead if the IoT devices need to switch from one network technology to another. *LoRa* allows activation by two methods Over-the-Air Activation (OTAA) and Activation by Personalisation (ABP). OTAA took on an average 5.6 s with a minimum 4 s to a maximum 7 s to join the network. On the other hand, ABP provides hard-coded session keys and allows the sending of data without joining. In case of an emergency where the device tries to send only one message and is unsure about the coverage of LoRa, the message can be sent using maximum SF12 to have a maximum range. For *Sigfox*, creating a socket for Sigfox taken an average of few ms. For NB-IoT, we present the timings for the different methods in Table VII.

When the LTE modem is connected to the network first time, it takes a significant amount of time to connect to the network as it searches, registers itself to the network, it could take approximately 15 $mins$ to 60 $mins$ to attach to the network, whereas Wi-Fi takes approx 5.6 seconds to connect to the specified network.

We conducted a baseline experiment for Lora to understand the connection time an end device takes to join the LoRa network through OTAA and repeated it 24 times. For NB-IoT, we conducted experiments to measure the time taken for initialisation, attach, connect, detach, disconnect, deinit and modem reset. We performed two experiments - one when

TABLE VII
NB-IoT connect times in seconds

NB-IoT		Reset	Init	Attach	Connect	Disconnect	Deattach	Deinit
With Reset	Avg	6.37	0	12.64	1.29	7.23	1.13	0.09
	Min	6	0	11	1	7	0	0
	Max	7	0	17	2	8	2	1
Without Reset	Avg	-	2.07	12.15	1.31	7.23	0.98	0.15
	Min	-	1	8	1	7	0	0
	Max	-	3	20	2	8	2	1

LTE modem is reset before initialisation and one without the reset. LTE allows PSM by configuring the period how often the device will connect and how long it will stay actively connected. During the sleep, the LTE modem will go into a low power state, but it will stay attached to the network; thus, no time is spent for attaching after waking up. For *Wi-Fi*, to understand the time taken to connect to Wi-Fi, we calculated the time taken for Wi-Fi init, scan, connect, disconnect, deinit. We experimented 80 times and found that it takes approximately 2.1 seconds to scan the Wi-Fi networks and approximately 5.6 seconds to connect to the specified network. Wi-Fi init, disconnect, and deinit were almost instantaneously in the range of milliseconds.

Time to reconnect to Wi-Fi, Internet: To understand how much time an IoT device takes to reconnect with the Wi-Fi and the internet. We connected a Smart Citizen Kit (SCK) [42], Pycom FiPy to Home Wi-Fi, a machine via Ethernet to the home router and turn-off-on the Wi-Fi. We created a python script that pings the three hosts: the router, the IoT device (SCK, FiPy), and the cloud machine (google.com) and provided time between the device going offline and coming online. It took approx 1 min 16 sec, 1 min 40 sec, 3 min 5 seconds to get the connectivity back to the router, IoT device, and the internet.

VI. Evaluation and Discussions

In § IV, we have shown that both variants of the proposed resource management algorithm ($CABF$ and $CABF_{inv}$) perform better than the baseline bin-packing algorithms we considered. In this section, we implemented one of the variants, namely $CABF_{inv}$, as part of a multi-network resource allocator running over our *Resilient Edge* platform. We then performed a number of experiments to evaluate the algorithm performance over an edge-node prototype following the experiment setup as shown in the Figures 4 and 5. The choice of the $CABF_{inv}$ was made in order to try to provide service to all message flows (rather than focus on increasing service for the most critical flows, which would perhaps be the goal in a real deployment) for the sake of demonstrability, i.e. so we can show the sharing of the network interfaces by several flows operating at different levels of criticality.

A. Criticality-aware allocation of network resources using $CABF_{inv}$

In this section, we show the performance of the proposed $CABF_{inv}$ algorithm when allocating network resources to application flows in a criticality-aware manner. We consider the same application flows and QoS requirements presented in Section § III and follow the approach described in Section § IV where application flows can request service at different criticality levels from the multi-network resource allocator running at FiPy. The multi-network resource allocator running $CABF_{inv}$ algorithm provides service according to those requirements while considering the network availability conditions, so in this experiment we consider a number of realistic scenarios and evaluate the percentage of served requests and the corresponding criticality levels they were assigned.

TABLE VIII
Obtained criticality level (1 | 2 | 3) and network allocation (* Wi-Fi | # LoRa | + Sigfox | - NB-IoT) for motivating example in FiPy

	Message Flows								% flows served	avg crit level
	1	2	3	4	5	6	7	8		
Requested Criticality level	1,2,3	1,2,3	1,2	1,2	1,2	1,2	1,2	1		
Network Interfaces	**Allocated Criticality Level**									
Wi-Fi	1	1	1	1	1	1	1	1	100	1
LoRa	1	1	1	2	1	2	1	1	100	1.25
NB-IoT	1	1	1	1	1	1	1	1	100	1
Sigfox	2	2	1	2	1	2	2	1	100	1.625
Wi-Fi + LoRa	1#	1#	1#	1*	1#	1*	1#	1#	100	1
Wi-Fi + Sigfox	1#	1#	1+	1#	1+	1#	1#	1+	100	1
NB-IoT + LoRa	1*	1*	1*	1-	1*	1-	1*	1*	100	1
NB-IoT + Sigfox	1-	1-	1+	1-	1+	1-	1-	1+	100	1

In this experiment we consider the four available networks have following maximum bandwidths Wi-Fi (750 $Kbps$), NB-IoT UL (55 $Kbps$), LoRa SF7-125KHz (5.47 $Kbps$) and Sigfox UL (100 bps).

Table VIII shows the allocated criticality level, percentage of flows served and average criticality level for the messages flows defined in Table I. Each row of the table shows the metrics obtained by running the $CABF_{inv}$ algorithm over a different network scenario. Scenarios include situations such as when only a single network is available (only Wi-Fi, LoRa, NB-IoT or Sigfox) or when two different networks are available (such as Wi-Fi or NB-IoT with LoRa and Sigfox). When a high-bandwidth network such as Wi-Fi and NB-IoT is available, we can see that $CABF_{inv}$ is able to assign the lowest criticality level to all flows and to provide all of them with service. We also observe that when only low-bandwidth network interfaces are available (e.g. Sigfox), all flows are still serviced but the average allocated criticality is higher (i.e. flows are only allowed to use the network under more constrained levels of service). Average criticality level is calculated as the sum of all the assigned criticality level divided by the number of message flow allocated.

Such results, which are based on a realistic scenario and network bandwidths, consistently show the same outcomes that were obtained in Section IV for our motivating example and for the synthetic applications: the proposed algorithms are superior to all baselines when one considers together the ability to allocate bandwidth to message flows according to their criticality and to the availability of multiple networks. There are, of course, limitations with regard to the performance of the proposed algorithms, our ability to fully exploit its advantages over the current platform, and the algorithms' ability to handle highly dynamic scenarios. We provide more details and discussion in the following subsections.

B. Time complexity and Context Switching of $CABF_{inv}$ algorithm

We measured the running time of $CABF_{inv}$ algorithm on the FiPy board and the RPi, repeated it ten times, and average run on FiPy takes $1300ms$ whereas on RPi it takes $7.1ms$.

When a networking event (such as Wi-Fi is disconnected), the allocation algorithm $CABF_{inv}$ has to be executed again. This results in time delay due to de-allocation of old message flows and allocation of new message flows. During this time delay, there's a possibility that the RPi would have written a message on the UART.

As there is a possibility that by the time, Message Flow Element Allocation (MFEA) message was received by RPi, the previous running threads (simulating message flows) would have written few messages to the UART. To resolve this, before doing the re-allocation, we send a message `<INFO:RE-ALLOC:INIT>` to RPi that, we are going to do the re-allocation, stop sending any message to the UART to minimise the loss of messages. On receiving that message, RPi pauses all the current threads of message flow. Further, FiPy store the old allocations and until it receives an acknowledgement message `<INFO:RE-ALLOC:ACCEPTED>` from the RPi that it has received the MFEA, it keeps allocating using previous allocation (except the network interface which was lost).

In this case, we log the time, when `RE-ALLOC:INIT` message was written to the UART by FiPy initiating re-allocation, the time RPi received MFEA message flow allocation message from FiPy, and the time taken by RPi to stop all previous threads (which are simulating the message flows) and generate new threads (as per new allocation). We calculated the time for context switching as the time difference between re-allocation `init` message written by FiPy and the re-allocation accept message received by FiPy. This whole context switching takes $1.3\ s$ to $1.5\ s$ which includes stopping thread, creating new threads, re-alloc `init` message, re-alloc MFEA time from RPi to FiPy and re-alloc accept from FiPy to RPi.

C. Discussions

Our work also has certain limitations. Firstly, the CABF algorithm currently does not handle network dynamics such as a change in network bandwidth due to dynamic change of wireless channel and link conditions. The preliminary decision about the network capacity is based on the network availability (whether the network is available or not), and the algorithm calculates the network bandwidth at the start of the network connection. Currently, it is difficult to generate or simulate network problems during application communication to evaluate the consequences on the flows (latency, loss, throughput). For instance, currently, FiPy does not provide the Wi-Fi callback function [43] and does not provide any way to know that Wi-Fi is disconnected. In LTE, we can remove the SIM card or the LTE antenna during a stable connection to simulate network connectivity loss. However, removing SIM or antenna is not officially recommended as they can cause damage to the device. Regarding generating network loss in Lora and Sigfox, both are stateless. FiPy provides a way to check if the device has joined LoRaWAN; however, no way to find whether it is still connected or not. Because of the above reasons, to simulate the loss of Wi-Fi, we have manually set the Wi-Fi bandwidth to zero and then called the re-allocation function. The multi-network resource allocator successfully allocates the message flows to the available network interfaces. From the network bandwidth perspective (change in bandwidth due to network conditions), a for loop that checks for the LoRa SF, Wi-Fi, and NB-IoT bandwidth at regular intervals can be implemented. However, it requires better support for threading. We will eventually implement the features based on the device support for Wi-Fi callback in the future.

Secondly, there are few device limitations. FiPy does not provide Wi-Fi callback to indicate if the device got disconnected from the Wi-Fi network. Currently, when FiPy is connected to both Wi-Fi and NB-IoT simultaneously, it does not provide a way to define the network interface to be used for sending the packet. Further FiPy team does not advise using both networks simultaneously to simulate a WiFi-LTE bridge, as it will be very slow and expensive [26].

Thirdly, currently, we take a set of message flows and allocate them all together. Because of this, old message flows are de-allocated and re-assigned with either the same or different criticality levels. In future work, we will provide the capability to allow an application to define a new message flow and allocate it from the existing networks without de-allocating and re-allocating the old ones.

Further, there are different industrial products [44], [45] in the market that provide communication via multiple radio interfaces (such as Wi-Fi, 4G, LoRa, LTE (CAT-M1/NB-IoT)). However, either they provide only LoRa or LTE (CAT-M1/NB-IoT) with Wi-Fi. Currently, we are only aware of FiPy that provides multi-network connectivity for LoRa, LTE (CAT-M1/NB-IoT), Sigfox, Wi-Fi, and Bluetooth. Further, our work enables criticality-aware applications to send messages by allocating resources (network) per the criticality level and network availability. The transmission range of Wi-Fi and other WPAN is different, and it is possible to assign the communication resources to different types of traffic. There can be different factors for consideration in the case of multiple radio devices, e.g., bandwidth, delay, rate adaptation, IP support, and others. Currently, our work considers bandwidth and availability to ensure that applications can send messages as per the defined criticality level.

With the development and popularization of 4G/5G networks, the IoT edge has also shown more possibilities in IoT, VR, and AI intelligence. In this context, NB-IoT, LoRa, and Sigfox provide low-bandwidth network communication methods that are very limited. There might be a case where LPWAN might seem insignificant. On the other hand, our work targets critical edge applications that need to work even when high-bandwidth networks are unavailable.

VII. Related Work

This section presents related work that crosses the intersection of LPWAN, edge resilience, and ILP (Integer Linear Programming) formulations for IoT and edge computing.

Chaudhari et al. [46] provided a comprehensive survey on various LPWAN technologies and presented these technologies concerning application requirements, such as coverage, capacity, cost, low power, and deployment complexity, and provided a comprehensive survey on both standard and non-standard LPWAN technologies. Hossain et al. [47] presented the comparison of different LPWAN technologies in terms of cost structure and scalability and stated that a large rollout with a single LPWAN technology is not cost-efficient.

Similarly, from the use case perspective, Santos et al. [48] evaluated LPWAN technologies for air quality application during "City of Things" project. Further, it also performed anomaly detection for smart city applications using different unsupervised and outlier detection algorithms. Roque et al. [49] created a prototype to detect fire detection in outdoor environments (forests) based on LPWAN networks (Sigfox) and temperature and gas sensor measurements. Rubio-Aparicio et al. [5] implemented an LPWAN residential water management solution supported by hybrid IoT LoRa-Sigfox architecture. All the above solutions provide resiliency by sending data on Lora and Sigfox without guaranteeing applications' QoS requirements. The work aims to achieve network resiliency by connecting the end devices with inadequate coverage to a Lora-Sigfox Gateway device via LoRa and then forwarding the data to a Sigfox network. These use-uses demonstrate the use of LPWAN for meeting resiliency and low-power communication requirements.

ILP formulations for resource provisioning are widely used for many scheduling problems and are well studied in the literature. For IoT applications, ILP has been used at the gateway level. For example, Santos et al. [50] presents a MILP (Mixed ILP) formulation for resource provisioning in Fog computing, taking into account the Service Function Chaining (SFC) concepts, different LPWAN technologies (LoRa, IEEE 802.11 ah), and multiple optimization objectives. The solution considers end-to-end systems into three segments - sensors/things level, gateways/routers (Fog), and the cloud and presents smart-city use-cases for garbage collection, air quality monitoring, and closed-circuit television (CCTV) monitoring. Tajiki et al. [51] used ILP to select a set of monitoring flow injected into the network to infer a link delay vector and meet the QoS for delay-sensitive applications in the network. Kim et al. [52] use ILP formulation to create secure migration policies for the communication between things (sensors) and a trusted edge system providing authentication services in the event of Denial-of-Service (DoS) attacks or failures, resulting in resilient authentication and authorization for IoT. In comparison, our work shows that IPL can also be used at the IoT device level to optimize the latency and resiliency of different applications using a Multi-communication network.

From the QoS perspective, multiple research papers have highlighted that the end-to-end perceived QoS on cloud-edge continuum deployment environments depends on many complex system factors [53], [54], [55].

Each abstraction (either vertical or horizontal) adds another level of complexity and delays, affecting QoS. The delays can depend on each edge node's virtualization and containerization techniques [56]. Additionally, many QoS (latency and processing delays) metrics depend on the current load of the local physical/virtual CPU/memory, network acceleration and service invocation techniques [56], [57].

For example, Cicconetti et al. [54] identified four reference execution models (external, in-edge, in-function, in-client) for providing state to enable stateful applications on serverless platforms deployed on the edge nodes. Similarly, Pfandzelter et al. [58] and Feraudo et al. [59] designed a lightweight serverless platform, tinyFaaS and Colearn middleware explicitly for edge environments and IoT applications.

From the literature, it is evident that edge-enabled FaaS scenarios with serverless support are emerging, and our work is complementary to state-of-the-art work. It can be integrated with middleware or service orchestration architecture by interfacing the Resilient Edge at the communication layer interface or as the network functions virtualization (NFV) in Software-Defined Networking (SDN) architecture.

From the perspective of improving resilience, Qin et al. implemented Multinetwork INformation Architecture (MINA) [60], [61] a reflective Observe Analyse Adapt (OAA) middleware approach to manage dynamic and heterogeneous multi-network (such as ZigBee, Bluetooth, PANs, MANETs, 3G/4G, WLAN) in pervasive environments to ensure reliable communication for end applications. The paper presented a formal analysis that can guide network administrators in their decisions to proactively adapt network configurations to achieve mission or application objectives. Compared to this work in our paper, we analyzed seamless switching of the networks on a hardware testbed to meet the resiliency requirements. In our prototype we provided seamless switching while maintaining the critical application requirements without overhead of virtualization and service orchestration middleware. However, our solution could be easily integrated to other intermediate middleware to support application critical requirements while providing seamless network connectivity.

The SCALE2 [62] leveraged MINA and implemented a multi-tier and multi-network approach to drive data flow from IoT devices to cloud platforms. The authors implemented a local Software-defined networking (SDN)-enabled the network, which is adaptive to the network changes to which IoT client devices are connected. This solution's architecture and deployment examples used separate adapters (device) for each communication radio, thus needing another computing device to run the SCALE client software. However, in our work, we use all radios integrated on a single board to allow fast switching between networks on the device level.

Wider aspects of resilience have been discussed in mission-critical applications like autonomous driving, tactile healthcare, and public safety. For example, Modarresi et al. [63] presented a graph-theoretical approach to model IoT systems in smart homes with integrated heterogeneous networks and explored resilience properties. Similarly, Chaterji et al. [64] presents the resilience of Cyber Physical System (CPS) and discusses two techniques resilience-by-design and resilience-by-reaction. Harchol et al. [65] proposed a framework to improve edge-computing resilience for session-oriented applications. They utilized message replay and checkpoint-based mechanisms to make client-edge-server systems more tolerant

to edge failures and client mobility. Carvalho et al. [66] implement a replication mechanism LoRa-REP for replicating critical messages on LoraWAN by sending them at different SF and improving redundancy in LoRaWAN for mixed-criticality scenarios.

The literature shows that different forms of replication and redundancy mechanisms are used to achieve resilience in the networks. However, none of those mentioned above work used different LPWAN and Wi-Fi as seamless multi-network infrastructure at the device and the Edge network to meet the guaranteed message delivery.

Our work focuses on achieving network resiliency using the LPWAN network on resource-constrained end devices by providing the capability to the end device to evaluate the application requirements and select the suitable network medium while allowing graceful degradation of services in the event of failures. Further, our work implements an ILP solver in micro-python that can run on a resource-constrained device. Also, multi-network connectivity has benefits in terms of deployment in mission-critical applications (tactile healthcare, public safety in smart cities). For mobility-based IoT like autonomous driving, for example, if one type of network exists in one area. In contrast, there is another network in another geographical location, and the application can perform smooth and seamless network switching.

VIII. Conclusion and Future Work

The resiliency and reliability requirements of IoT applications vary from non-critical (best delivery efforts) to safety-critical with time-bounded guarantees. In this work, we systematically investigated how to meet these applications mixed-criticality QoS requirements in multi-communication networks.

We presented the network resiliency requirements of IoT applications by defining a theoretical multi-network resource system model and proposed and evaluated a list of resource allocation algorithms and found Criticality-Aware Best Fit ($CABF_{inv}$) algorithm works better to meet high criticality requirements of the example applications. The algorithm provides the best-effort QoS match by taking into consideration the underlying dynamic multi-network environments. We analysed and evaluated the bandwidth, latency, throughput, maximum packet size of LPWAN technologies, such as Sigfox, LoRa, and NB-IoT and implemented and evaluated an adaptive *Resilient Edge* system with Criticality-Aware Best Fit (CABF) resource allocation to meet the application resiliency requirements using underlying LPWAN technologies on RPi and FiPy.

In the current implementation of *Resilient Edge*, we took bandwidth and subsequent inter-message period into consideration for defining criticality [6]. In future, we would like to extend multi-network resource allocator to include message payload size, message transmission frequency, security, privacy and energy consumption parameters in the allocation algorithm. The new allocator would provide applications more flexibility to choose and optimise their resources and QoS for a multi-communication network. In summary, we investigated the limits and metrics required for the best-effort high criticality resilience in multi-communication networks. We presented our findings on how to achieve 100% of the best-effort high criticality level message delivery using multi-communication networks. Our work will help build reliable applications on IoT Edge and provide solutions from the perspective of communication networks to improve service quality and fault tolerance on resource-constrained edge devices. It also opens up new research directions to build reliable and trustworthy IoT applications over robust and resilient IoT Edge.

[6]Github repository:

$https://github.com/pooyadav/smartcity\_multimode\_networks$

Acknowledgements

Yadav and Kumar are supported in part by "Data Negotiability in Multi-Mode Communication Networks" project funded by EPSRC grant EP/R045178/1. Kumar's PhD is supported by EPSRC studentship award EP/R511857/1 (2128072). For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

References

[1] Perfect Patient Pathway, "Perfect Patient Pathway Test Bed Overview Report 2," 2016. [Online]. Available: https://devicesfordignity.org.uk/wp-content/uploads/2019/03/Test-Bed-Overview-Report-DIGITAL-VERSION.pdf

[2] U. Ann Raymond, Sachin Vadgama, Sonya Crowe, Steve Morris and Martin Utley. (2019) Evaluation of the nhs england innovation test bed at care city. Accessed:2020-06-30. [Online]. Available: https://www.carecity.london/publications/reports/15-evaluation-of-the-nhs-england-innovation-test-bed-at-care-city/file

[3] P. Yadav, V. Safronov, and R. Mortier, "Enforcing accountability in smart built-in iot environment using mud," in *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, ser. BuildSys'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 368–369. [Online]. Available: https://doi.org/10.1145/3360322.3361004

[4] M. Sweney. (2020) Bristol is worst uk city for broadband outages with 169 hours a year. Accessed:2020-06-30. [Online]. Available: https://www.theguardian.com/technology/2020/aug/13/bristol-is-worst-uk-city-for-broadband-outages-with-169-hours-a-year

[5] J. Rubio-Aparicio, F. Cerdan-Cartagena, J. Suardiaz-Muro, and J. Ybarra-Moreno, "Design and implementation of a mixed IoT LPWAN network architecture," *Sensors (Switzerland)*, vol. 19, no. 3, 2019.

[6] V. Kumar, P. Yadav, and L. S. Indrusiak, "Poster: Building iot resilient edge using lpwan and wifi," in *ACM IMC*, 2021.

[7] J. Moore, A. Arcia-Moret, P. Yadav, R. Mortier, A. Brown, D. McAuley, A. Crabtree, C. Greenhalgh, H. Haddadi, and Y. Amar, "Zest: Rest over zeromq," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2019, pp. 1015–1019. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/PERCOMW.2019.8730686

[8] Pycom, "Fipy experiment board," 2020, accessed: 2020-06-30.

[9] Semtech, "LoRa and LoRaWAN," *Semtech Technique Paper*, no. December 2019, pp. 1–17, 2020. [Online]. Available: https://lora-developers.semtech.com/library/tech-papers-and-guides/lora-and-lorawan/

[10] A. Technologies. (2020) Robeau lorawan water flow meter. Accessed:2020-06-30. [Online]. Available: https://www.alliot.co.uk/products/sensors/water-sensors/robeau-lorawan-water-flow-meter/

[11] LoRa Alliance Technical Commitee, "LoRaWAN 1.1 Specification," *LoRaWAN 1.1 Specification*, no. 1.1, p. 101, 2017. [Online]. Available: https://lora-alliance.org/resource-hub/lorawantm-specification-v11

[12] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of lpwan technologies for large-scale iot deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959517302953

[13] T. M. Workgroup, "A technical overview of LoRa® and LoRaWAN," no. November, 2015. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf

[14] L. D. Portal. (2020) An in-depth look at lorawan® class a devices. Accessed:2020-06-30. [Online]. Available: https://lora-developers.semtech.com/library/tech-papers-and-guides/lorawan-class-a-devices

[15] Sigfox, "Letterbox sensor," https://partners.sigfox.com/products/letterbox-sensor, 2020, accessed:2020-06-30.

[16] ——, "Sigfox Technical Overview," vol. 1, no. May, p. 26, 2017. [Online]. Available: https://www.disk91.com/wp-content/uploads/2017/05/4967675830228422064.pdf

[17] GSMA. (2020) Lte-m deployment guide to basic feature set requirements. Accessed:2020-06-30. [Online]. Available: https://www.gsma.com/newsroom/wp-content/uploads//CLP.29-v2.0.pdf

[18] S. Øyvann, "Internet of sheep: Why world's biggest nb-iot pilot is roping in woolly helpers," https://www.zdnet.com/article/internet-of-sheep-why-worlds-biggest-nb-iot-pilot-is-roping-in-woolly-helpers/, 2020, accessed:2020-06-30.

[19] Arkessa. (2019) Cellular iot solution guide. Accessed:2020-06-30. [Online]. Available: https://www.arkessa.com/?download_id=1972&p_id=1971&type=latest

[20] GSM Association, "LTE-M Commercialisation. Case study how AT&T and Telstra connect million more IoT devices." pp. 1–10, 2019.

[21] GSMA, "NB-IoT Deployment guide to basic feature set requirements," *Gsma*, vol. Release 3, no. June, pp. 1–80, 2019. [Online]. Available: https://www.gsma.com/iot/resources/nbiot-deployment-guide-v3/

[22] J. Harbin, A. Burns, R. I. Davis, L. S. Indrusiak, I. Bate, and D. Griffin, "The AirTight Protocol for Mixed Criticality Wireless CPS," *ACM Trans. Cyber-Phys. Syst.*, vol. 4, no. 2, 2019. [Online]. Available: https://doi.org/10.1145/3362987

[23] I. Correia, L. Gouveia, and F. Saldanha-da Gama, "Solving the variable size bin packing problem with discretized formulations," *Computers & OR*, vol. 35, pp. 2103–2113, 06 2008.

[24] R. Pi. (2020) Raspberry pi 4 model. Accessed:2020-06-30. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

[25] Pycom, "Fipy datasheet," https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_FiPy_v2.pdf, 2020, accessed:2020-06-30.

[26] "Fipy: How to send data via simultaneous connected with both wi-fi and lte (nb-iot/ cat-m)," https://forum.pycom.io/topic/6204/fipy-how-to-send-data-via-simultaneous-connected-with-both-wi-fi-and-lte-nb-iot-cat-m, 2020, 2020-06-30.

[27] LoRa™ Alliance, "LoRaWAN™ Regional Parameters v1.1rA," *LoRaWAN™ 1.1 Specif.*, p. 56, 2017. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-05/lorawan-regional-parameters-v1.1ra.pdf

[28] A. van Bentem, "Airtime calculator for lorawan," https://avbentem.github.io/airtime-calculator/ttn/eu868, 2020, 2020-06-30.

[29] T. T. Network. (2020) Lorawan airtime calculator. Accessed:2020-06-30. [Online]. Available: https://www.thethingsnetwork.org/airtime-calculator

[30] ——. (2020) Duty cycle. Accessed:2020-06-30. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html

[31] ——. (2020) Best practices to limit application payloads. Accessed:2020-06-30. [Online]. Available: https://www.thethingsnetwork.org/forum/t/best-practices-to-limit-application-payloads/1302/1

[32] S. Support. (2020) Link quality: general knowledge. Accessed:2020-06-30. [Online]. Available: https://support.sigfox.com/docs/link-quality:-general-knowledge

[33] ——. (2020) A technicality: padding bits. Accessed:2020-06-30. [Online]. Available: https://build.sigfox.com/payload#a-technicality-padding-bits

[34] G. R. L. Bras. (2020) What is the difference in data throughput between lte-m/nb-iot and 3g or 4g? Accessed:2020-06-30. [Online]. Available: https://www.gsma.com/iot/resources/what-is-the-difference-in-data-throughput-between-lte-m-nb-iot-and-3g-or-4g/

[35] PyCom. (2020) Vodafone nb-iot prepaid subscription. Accessed:2020-06-30. [Online]. Available: https://pycom.io/product/vodafone-nb-iot-prepaid-subscription

[36] shawwwn — Github Gist. (2020) μping: Ping library for micropython. Accessed:2020-06-30. [Online]. Available: https://gist.github.com/shawwwn/91cc8979e33e82af6d99ec34c38195fb

[37] shawwwn. (2020) uping usage - micropython forum. Accessed:2020-06-30. [Online]. Available: https://forum.micropython.org/viewtopic.php?f=15&t=5287

[38] T. T. Network. (2020) Frequency plans. Accessed:2020-06-30. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/frequency-plans.html

[39] S. Build, "Radio configurations : Rc technical details," https://build.sigfox.com/sigfox-radio-configurations-rc, 2020, accessed:2020-06-30.

[40] E.-I. P. Guide. (2020) Esp32 wi-fi throughput. Accessed:2020-06-30. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html#esp32-wi-fi-throughput

[41] D. George. (2020) uiperf3 — pure python, iperf3-compatible network performance test tool. Accessed:2020-06-30. [Online]. Available: https://pypi.org/project/uiperf3/

[42] S. C. Kit, "Smart citizen kit docs," https://docs.smartcitizen.me/Smart%20Citizen%20Kit/, 2020, accessed:2020-06-30.

[43] "Event based programming/ interrupts — fipy — network (wifi/lte/lora) disconnection," https://forum.pycom.io/topic/6341/event-based-programming-interrupts-fipy-network-wifi-lte-lora-disconnection, 2020, 2020-06-30.

[44] TELTONIKA, "Teltonika industrial cellular modem with multiple lpwan connectivity options (trm250)," https://www.wifi-stock.com/details/teltonika-industrial-cellular-modem-trm250.html, 2020, accessed:2020-06-30.

[45] MultiTech, "Multitech brochure product lpwa solutions for industrial iot," https://www.multitech.com/documents/publications/brochures/MT_Brochure_Product_LPWA_2019-10-15.pdf, 2020, accessed:2020-06-30.

[46] B. S. Chaudhari, M. Zennaro, and S. Borkar, "Lpwan technologies: Emerging application characteristics, requirements, and design considerations," *Future Internet*, vol. 12, no. 3, 2020. [Online]. Available: https://www.mdpi.com/1999-5903/12/3/46

[47] M. I. Hossain and J. I. Markendahl, "Comparison of lpwan technologies: Cost structure and scalability," *Wireless Personal Communications*, vol. 121, no. 01, pp. 887–903, Nov 2021.

[48] J. Santos, P. Leroux, T. Wauters, B. Volckaert, and F. De Turck, "Anomaly detection for Smart City applications over 5G low power wide area networks," *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, pp. 1–9, 2018.

[49] G. Roque and V. S. Padilla, "LPWAN Based IoT Surveillance System for Outdoor Fire Detection," *IEEE Access*, vol. 8, pp. 114 900–114 909, 2020.

[50] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards end-to-end resource provisioning in Fog Computing over Low Power Wide Area Networks," *Journal of Network and Computer Applications*, vol. 175, no. August 2020, 2021.

[51] M. M. Tajiki, S. H. G. Petroudi, S. Salsano, S. Uhlig, and I. Castro, "Optimal estimation of link delays based on end-to-end active measurements," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4730–4743, 2021.

[52] H. Kim, E. Kang, D. Broman, and E. A. Lee, "Resilient Authentication and Authorization for the Internet of Things (IoT) Using Edge Computing," *ACM Transactions on Internet of Things*, vol. 1, no. 1, pp. 1–27, 2020.

[53] A. Orive, A. Agirre, H.-L. Truong, I. Sarachaga, and M. Marcos, "Quality of service aware orchestration for cloud-edge continuum applications," *Sensors (Basel)*, vol. 22, no. 5, Feb 2022.

[54] C. Cicconetti, M. Conti, and A. Passarella, "In-network computing with function as a service at the edge," *Computer*, vol. 55, no. 09, pp. 65–73, sep 2022.

[55] A. Garbugli, A. Sabbioni, A. Corradi, and P. Bellavista, "Tempos: Qos management middleware for edge cloud computing faas in the internet of things," *IEEE Access*, vol. 10, pp. 49 114–49 127, 2022.

[56] S. Guo, K. Zhang, B. Gong, W. He, and X. Qiu, "A delay-sensitive resource allocation algorithm for container cluster in edge computing environment," *Computer Communications*, vol. 170, pp. 144–150, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366421000414

[57] I. Pelle, M. Szalay, J. Czentye, B. Sonkoly, and L. Toka, "Cost and latency optimized edge computing platform," *Electronics*, vol. 11, no. 4, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/4/561

[58] T. Pfandzelter and D. Bermbach, "tinyfaas: A lightweight faas platform for edge environments," in *2020 IEEE International Conference on Fog Computing (ICFC)*, 2020, pp. 17–24.

[59] A. Feraudo, P. Yadav, V. Safronov, D. A. Popescu, R. Mortier, S. Wang, P. Bellavista, and J. Crowcroft, "Colearn: Enabling federated learning in mud-compliant iot edge networks," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 25–30. [Online]. Available: https://doi.org/10.1145/3378679.3394528

[60] Z. Qin, L. Iannario, C. Giannelli, P. Bellavista, G. Denker, and N. Venkatasubramanian, "MINA: A reflective middleware for managing dynamic multinetwork environments," *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, pp. 3–6, 2014.

[61] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-of-things," *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, 2014.

[62] M. Y. S. Uddin, A. Nelson, K. Benson, G. Wang, Q. Zhu, Q. Han, N. Alhassoun, P. Chakravarthi, J. Stamatakis, D. Hoffman, L. Darcy, and N. Venkatasubramanian, "The Scale2 Multi-Network Architecture for IoT-Based Resilient Communities," *2016 IEEE International Conference on Smart Computing, SMARTCOMP 2016*, 2016.

[63] A. Modarresi and J. Symons, "Resilience and technological diversity in smart homes: A graph-theoretic approach to modeling IoT systems with integrated heterogeneous networks," *Journal of Ambient Intelligence and Humanized Computing*, no. Alliance 2008, 2020. [Online]. Available: https://doi.org/10.1007/s12652-020-02095-8

[64] S. Chaterji, P. Naghizadeh, M. A. Alam, S. Bagchi, M. Chiang, D. Corman, B. Henz, S. Jana, N. Li, S. Mou, M. Oishi, C. Peng, T. Rompf, A. Sabharwal, S. Sundaram, J. Weimer, and J. Weller, "Resilient Cyberphysical Systems and their Application Drivers: A Technology Roadmap," pp. 1–36, 2019. [Online]. Available: http://arxiv.org/abs/2001.00090

[65] Y. Harchol, A. Mushtaq, V. Fang, J. McCauley, A. Panda, and S. Shenker, "Making edge-computing resilient," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 253–266. [Online]. Available: https://doi.org/10.1145/3419111.3421278

[66] D. F. Carvalho, P. Ferrari, E. Sisinni, and A. Flammini, "Improving Redundancy in LoRaWAN for Mixed-Criticality Scenarios," *IEEE Systems Journal*, pp. 1–10, 2020.

Vijay Kumar is a final year PhD Student at the University of Bristol, UK. Before joining his PhD, he worked as a senior cybersecurity consultant with an International Consultancy Firm. He received MSc in Advance Computing with a specialisation in Internet Technologies with Security from the University of Bristol, UK, in 2013. His research interests are in Smart Cities, the Internet of Things, Computer Forensics and Security.

Poonam Yadav is currently an Assistant Professor with the Computer Science Department, University of York, UK. She received the M.Tech. degree from the Indian Institute of Information Technology, Allahabad, India, in 2007, and the PhD degree in computing from Imperial College London, London, UK, in 2011. Her research interests include IoT applications, infrastructure, networking and medium access protocols, distributed and centralised architectures, examining resilience, security, privacy, latency, seamless integration, communication, computation and energy efficiency. Dr. Yadav was a recipient of the U.K.–India Education and Research Initiative Ph.D. Award. She was involved in various NERC, TSB, EU, EPSRC, IBM, and Microsoft-funded research projects. She is currently the Chair of ACM-W U.K. professional chapter and a senior member of ACM and BCS and is featured as "People of ACM Europe" and among the top ten N2Women Rising Star in computer networking and communications, in 2020.

Leandro Soares Indrusiak has been an academic with the Computer Science department at University of York since 2008, and currently serves as the leader of the Real-Time and Distributed Systems research group (RTDS) and director of the Doctoral Centre for Safe, Ethical and Secure Computing (SEtS). His main research interests include real-time systems and networks, evolutionary optimisation, on-chip multiprocessing, distributed embedded systems, and several types of resource allocation problems (in computing, manufacturing and transportation). He has published more than 150 peer-reviewed papers in the main international conferences and journals covering those topics, and has been the principal investigator on projects funded by the EU, EPSRC, DFG, British Council and industry.