2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)

Isolayer: The Case for an IoT Protocol Isolation Layer

Jiamei Lv, Gonglong Chen and Wei Dong*
College of Computer Science, Zhejiang University.
Alibaba-Zhejiang University Joint Institute of Frontier Technologies.
Email: {*lvjm, chengl, dongw*}*@emnets.org*

2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS) | 978-1-6654-1494-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/IWQOS52092.2021.9521313

***Abstract*—Internet of Things (IoT), which connects a large number of devices with wireless connectivity, has come into the spotlight. Various wireless radio technologies and application protocols are proposed. Due to scarce channel resources, different network traffic may do interact in negative ways. This paper argues that there should be an isolation layer in IoT network communication stacks making each traffic's perception of the wireless channel independent of what other traffic is running.**

We present Isolayer, an isolation layer design providing *fine-grained* and *flexible* channel isolation services in the *heterogeneous* IoT networks. By a shared collision avoidance module, Isolayer can provide effective isolation even between different wireless technologies (e.g., BLE and 802.15.4). Isolayer provides four levels of isolation services for users, i.e., protocol level, packet-type level and source-/destination-address level. Considering the various isolation requirements in practice, we design a domain-specific language for users to specify the key logic of their requirements. Taking the codes as input, Isolayer generates the control packets automatically and lets related nodes that receive the control packets update their isolation services correspondingly.

We implement Isolayer on realistic IoT nodes, i.e., TI CC2650, Heltec LoRa node 151, and perform extensive evaluations. The results show that: (1) Isolayer incurs acceptable overhead in terms of delay and memory usage; (2) Isolayer provides effective isolation service in the heterogeneous IoT network. (3) Isolayer achieves about 18.6% reduction of the end-to-end delay of isolated packets in the IoT network with heavy traffic load.

I. Introduction

Recent years have witnessed the rapid growth of IoT (Internet of Things) technologies. Various wireless radio technologies and application protocols have been proposed to meet the various requirements. When multiple protocols are running simultaneously, it may cause a network-wide failure due to the channel resource starvation from one protocol [1]. For example, when MQTT and CoAP coexist, the bursty transmissions of CoAP may cause a considerable delay in receiving or forwarding the control messages from MQTT, resulting in a crucial failure, especially for the delay-sensitive task.

There are a few works proposed to deal with the above problems. FWP [1], an isolation layer for sensor networks, sits between the MAC layer and the network layer. It provides fair resource scheduling between different protocols. However, the services provided by FWP are *insufficient* for the various requirements from the different applications in IoT networks.

- Wireless radio technologies are diverse in IoT protocols. Heterogeneous wireless radio technologies will exist in the same IoT network because of various application requirements (e.g., 802.15.4 for short-distance transmission, LoRa [2] for long-distance transmission). Recently, multi-standard radio technologies (e.g., TI CC2650 supports for BLE and 802.15.4 [3]) are proposed to enable communications among heterogeneous IoT devices in a single chip and have attracted many attentions in the industry. However, FWP only supports the single radio since its key modules are designed specifically for a single radio (e.g., GTS for collision avoidance).
- The fine-grained protocol control is desired in IoT networks. For example, in the smart home where an IoT system continuously monitors the home's safety (e.g., smoke, gas) and environment (e.g., heat, air, light), users expect that packets about safety (e.g., packets from smoke) will not be interfered by other packets [4]. FWP is single-rule and only provides isolation between different protocols. Therefore, it is incapable of such device-related isolation.
- The isolation requirements may change over time. For example, in the smart home scenario mentioned above, an IoT system continuously monitoring elderly health and alert emergency services in case of abnormal conditions is newly deployed. In this case, the alerts packets want to have a higher priority to use the channel which means the isolation rules need to be updated. However, FWP only provides the fixed isolation services unless reprogramming the IoT nodes.

In this paper, we propose an isolation layer design, Isolayer, which supports the *fine-grained* and *flexible* channel resources isolation in the *heterogeneous* IoT networks. Based on the analysis of different requirements of users from a large amount of IoT applications [5]–[7], Isolayer provides multi-level isolation, i.e., protocol level (e.g., MQTT vs. CoAP), packet-type level (e.g., ACK packets vs. data packets), source-/destination-address level according to the multi-dimension rules. Compared with FWP that only provides a single rule, Isolayer with a multi-dimensional rule engine can meet the various

*Corresponding author. This work is supported by the National Natural Science Foundation of China under Grant No. 62072396 and No. 61772465, and Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars under No. LR19F020001.

978-1-6654-1494-4/21/$31.00 ©2021 IEEE

requirements in IoT applications better. However, providing multi-level isolation will encounter many new challenges.

First, how to design the isolation layer to provide unified network control services and to be compatible with heterogeneous wireless radio technologies?

Solution: We design an isolation layer in existing IoT protocol stack which is decoupled from other layers. It sits upon the MAC layer and providing isolation service by packets classifier module, weighted fair queue-based packet queue, .etc. We design an adaptation module for incorporating different radio technologies. Considering the channel contention between the different wireless technologies, we design a shared collision avoidance building on cross-technology communication, providing the sender with clear channels even in a heterogeneous network.

Second, how to reduce the overhead introduced by the isolation layer?

Solution: To reduce the transmission overhead caused by complex operations from the multiple rules, the isolation layer is designed as condition-triggered. The nodes only provide effective isolation for packets of interest when needed. Besides, we take a series of measures to reduce its overhead, including transmitting packets strategically, employing a circular queue, etc.

Third, how to make the generation of the multi-dimension rules more user-centric?

Solution: We propose a domain-specific language (DSL) that encapsulates some implementation details (e.g., communication protocols, ports), making Isolayer user-friendly to non-experts. By parsing the control code written by users, Isolayer generates the control packets and distributes them to update or activate the isolation service of the networks automatically without having to re-burn the code to the IoT nodes, which is time- and labor-saving.

Isolayer is implemented on realistic IoT nodes based on the existing IoT network stack in RIOT OS [8]. We perform extensive experiments to evaluate it. The results show that: (1) Isolayer incurs acceptable overhead in execution time and program memory; (2) Isolayer provides effective and hierarchical isolation services in the heterogeneous IoT network; (3) Under heavy traffic loads, Isolayer reduces the end-to-end delay of the traffic of interest significantly, about 18.6% reduction.

Specifically, our contributions are as follows:

- We present a flexible isolation layer design, Isolayer, for IoT network which provides fine-grained and effective isolation services. We propose a shared collision avoidance mechanism making Isolayer compatible with different wireless radio technologies. To some extent, Isolayer laid the foundation for the promotion of multi-radio applications.
- We design a domain-specific language (DSL) for users to express their requirements without considering the networks' details. Isolayer automatically updates or activates the isolation services according to the control code written by users. By encapsulating some implementation details, Isolayer is user-friendly to non-experts.
- We implement Isolayer based on existing network stacks and evaluate it through a series of experiments. The results show that Isolayer can provide effective isolation according to the user requirements.

The rest of this paper is structured as follows. Section II gives two use cases of Isolayer. Section III is an overview of Isolayer, including the architecture of Isolayer and the module description. Section IV introduces the usage of Isolayer. Section IV describe the online isolation rules activation/update mechanism of Isolayer. Section VI introduces the shared collision avoidance mechanism while section VII introduce the isolation procedure. We evaluate Isolayer in Section VIII. Section IX is about the related work and section X concludes this paper.

II. Motivation

In this section, we study two real-world scenarios and use cases to illustrate the necessity for an isolation layer in the IoT network stacks.

DDos attack. With the explosive growth of IoT devices, they have been the primary force behind the biggest distributed denial of service (DDoS) botnet attacks for some time. Thousands of insecure IoT devices may be recruited to form botnets to attack other Internet users, and even used to attack critical national infrastructure, or the structural functions of the Internet itself [9]–[11]. It is a threat that has never really diminished, as numerous IoT device manufacturers continue to ship products that cannot be properly secured [12], [13].

Suppose the gateway, as the entrance of the network, can isolate different packets and only reserve most of the channel resources to the secure packets of interests, the gateway will be a security barrier that can prevent a potential network layer Dos attack [14] to some extent.

Industrial IoT. Various wireless radio technologies and application protocols playing different roles in the industrial Internet of things (IIoT). For example, the sensing data are uploaded to the gateway with a client-server mode protocol (e.g., CoAP [15]), which is suitable for the multi-point-to-point scenarios (e.g., data collection [16]). In order to receive the control messages, the sensors often rely on a publish-subscribe mode protocol (e.g., MQTT [17]) to subscribe to the topics that are related to them. It is suitable for the point-to-point/multi-point scenarios (e.g. device control [18]). However, the packets related to the control message compete with data packets for limited channel resources and may suffer unpredictable delays because of back-off, retransmit .etc, leading to adverse consequences.

It is necessary to introduce an isolation layer that can isolate the important packets (e.g., those related to the machine' control) from other packets and reserve a certain channel share for them even when the network is incredibly congested. With the isolation layer, users can roughly estimate the end-to-end delay of the control packets avoiding unintended adverse consequences.

In general, it is extremely beneficial to add an isolation layer to the existing IoT network architecture.

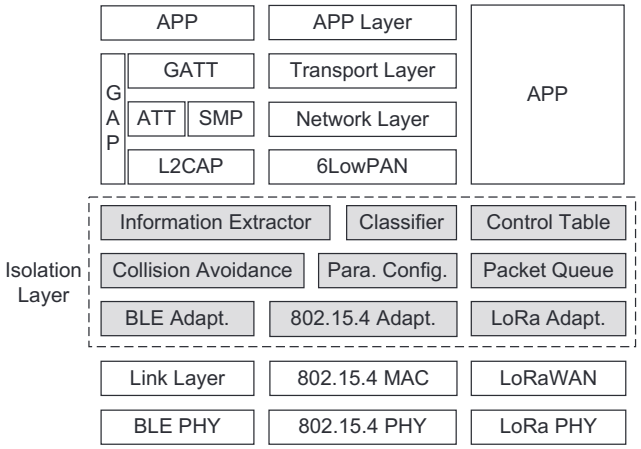


Fig. 1: Isolayer architecture overview. The gray boxes indicate newly implemented modules in Isolayer.

III. Isolayer overview

A. Architecture of Isolayer

Figure 1 shows the architecture of Isolayer where the gray boxes are the modules newly introduced. In the multi-radio nodes, it is a shared layer for different wireless technologies. Isolayer achieves the goal of *providing fine-grained and flexible channel resources isolation with low overhead in the heterogeneous IoT networks* by three mechanisms: online isolation service activation/update, shared collision avoidance mechanism and conditionally-triggered isolation. We sketch out these three mechanisms next.

1) Online isolation service activation/update: Unlike packets receiving, packets transmission, .etc, the isolation service is unnecessary in some scenarios, such as where traffic load is light. Therefore, Isolayer has two states: *unactivated* and *activated*. In *unactivated* state, the nodes implemented with Isolayer work like the nodes implemented with traditional network stack (e.g., GNRC in RIOT OS and uIP in Contiki) with low overhead. In *activated* state, the nodes provide the isolation service according to the users' requirements.

With online service activation/update, Isolayer can activate or update the isolation services without reprogramming the IoT nodes. First, the node implemented with Isolayer extracts the isolation rules from the incoming control packets. Next, if its isolation layer is activated, the node will update the control rules; otherwise, it activates the Isolation layer by writing the control rules into the control table. We carefully design the control package structure to reduce its transmission overhead which will be detailed in Section V.

2) Shared collision avoidance mechanism: Shared collision avoidance mechanism allows multiple packets of different protocols, wireless technologies or paths to avoid collisions with others. Isolayer uses existing CTC technologies, i.e., BlueBEE [19] and XBee [19], to achieve the communication between different technologies. Based on this, nodes implemented with Isolayer will send a little packet to silence the surrounding nodes, providing clear channels for the isolated packets. The mechanism will be introduced in detail in Section VI.

TABLE I: Mechanisms provided by, and modules implemented in Isolayer.

Mechanisms	Modules
Online isolation service activation/update	Classifier, Information extractor, Parameters configurator, Control table
Shared collision avoidance	Collision avoidance module, BLE adaptation, 802.15.4 adaptation
Conditionally-triggered isolation	Classifier, Control table, Packet queue

3) Conditionally-triggered isolation: The key of conditionally-triggered isolation is packet scheduling based on the weighted fair queue. Isolayer provides several weighted fair queues and configures their weights according to the control table. Once the isolation layer of the node is activated, the node identifies every outgoing packet. If the packet meets the conditions recorded on the control table indicating that it is a packet that needs to be isolated, Isolayer will queue it in the corresponding queue. We will elaborate on this mechanism in Section VII.

B. Module description

Table I summarizes the relationship between the mechanisms and the modules that provide them. We briefly describe these modules.

Classifier. There are generally two types of packets in Isolayer: normal packets and packets that need to be isolated. The classifier is responsible for identifying these two packets and directing them to the corresponding processing modules.

Information extractor. This module extracts the information in the control packets and reorganizes them into isolation rules.

Control table. The control table stores the isolation rules which will be updated whenever a new control packet is received. It is also an indicator of node states. If it is empty, the node is unactivated, otherwise activated.

Packet queue. The packet queue module performs buffer management and packet scheduling, which is a key module of Isolayer. Isolayer provides two types of packet queues: FIFO queue and weighted fairness queue (WFQ). The FIFO queue works when the node is unactivated while the WFQ works when activated.

Parameter configurator. This module configures the packet queues' parameters (e.g., the number and weights) to help provide different channel isolation services.

Collision avoidance. This module provides a shared collision avoidance mechanism based on CTC in the heterogeneous network, mainly for BLE and 802.15.4.

Adaption. Isolayer has three MAC adaptation modules: BLE adaptation, LoRaWAN adaptation, and 802.15.4 MAC adaptation. These modules encapsulate MAC layer functionalities and provide unified interfaces to the isolation layer.

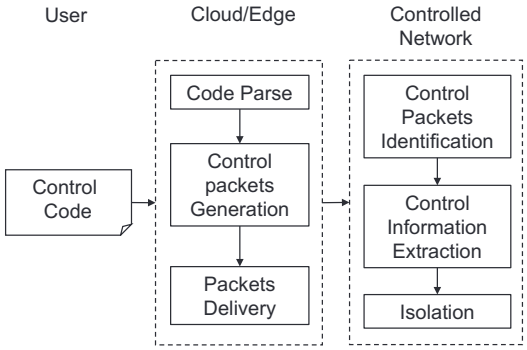


Fig. 2: Workflow overview of Isolayer.

```
1  Object o_mqtt;
2  o_ mqtt.protocol=MQTT;
3  share(o_mqtt)=0.3;
```

Fig. 3: An control code example of Isolayer. The meaning is that MQTT packets have 30% channel resources.

IV. Usage of Isolayer

Before introducing these three mechanisms mentioned above in detail, we present the usage of Isolayer firstly in this section for ease of explanation.

Figure 2 shows the workflow of Isolayer. First, users write the control code and upload it to the cloud/edge. The cloud/edge parses the code and then generates control packets sent to the controlled IoT network. Having received the control packets, the IoT nodes activate/update their isolation layer and provide isolation services.

As to the control code, we develop a domain-specific language (DSL) for users borrowing the program structure from C language which is a fairly popular and easy-to-learn programming language. Next, we describe the syntax details of control code by an example (see Figure 3) using Isolayer.

Writing a complete control code has the following three steps.

1) **Defining the isolated objects** (Line 1 in Figure 3). Users need to define the Isolated objects by using `Object` firstly. Users can define multiple objects in a control code simultaneously.
2) **Describing the objects** (Line 2 in Figure 3). Isolayer provides five attributes for users to describe the packets they want to isolate, as shown in Table II. It is worth noting that in order to make the system more practical, we design the reserved fields in the control packages (See Section V) so that the attributes can be extended easily.
3) **Allocating resources to the objects** (Line 3 in Figure 3). Isolayer provides one interface whose explanation is shown in Table III. Each defined object should be allocated resource. The parsing program will check the rationality of the data and prompt errors if there are any (e.g., more than 100% of the channel resources have been allocated.)

TABLE II: The attributes to describe the isolated objects.

Attributes	Description	Options
Radio	The radio technologies of the packets.	R_802154 (802.15.4 packets), BLE (BLE packets), LORA (LoRa packets) ...
Protocol	The APP layer protocols of the packets.	MQTT (MQTT packets), COAP (CoAP packets), ...
Pkt_type	The type of the packets	CONTROL (Control packets), DATA (Data packets), ...
Src_address	The source IP/MAC address of the packets.	—
Dest_address	The destination IP/MAC address of the packets.	—

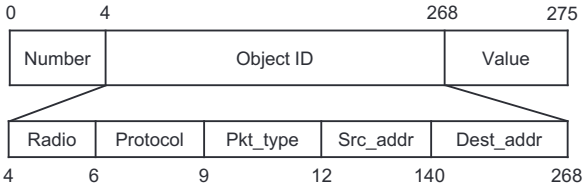


Fig. 4: An example of the data field of the control packet which has one object.

V. Online isolation service activation/update

A. Control packets generation.

A program running on the cloud/edge parses the control code uploaded by user and generates the corresponding control packets. Figure 4 shows an example of the control packets' data field. Every control packet carries three important information: number, object ID and value.

Number. The number indicates the number of isolated objects. Currently, Isolayer supports up to 16 isolated objects defined in one control code.

Object ID. The object ID helps identify the packets that should be isolated. It is a hierarchical format with five components which are `Radio`, `Protocol`, `Pkt_type`, `Src_addr` and `Dest_addr`, as same as the attributes mentioned in Section IV. The components not mentioned in the control code will be set to all 0s. It is worth emphasizing that `Src_addr` and `Dest_addr` fields sizes are 128 bits to support IPv6 addresses. In order to facilitate future expansion, the length of `Protocol` and `Pkt_type` are both 3 bits which means Isolayer can support up to 8 different protocols and types of packets, respectively.

Value. The value is the channel share the user wants to allocate to the isolated object, an integer between 1 and 100 taking up 7 bits. Every object define in the control code has the corresponding "Object ID" and "Value" fields in the control packet.

B. Activate/update isolation service

The IoT nodes will activate or update their isolation services once receiving the control packets. Each node has a control table, as shown in Table V which records the control rules extracted from control packets, including object ID and the corresponding channel share. If the control table is empty, the node is unactivated originally. The nodes will write the rules extracted to the table and switch to the activated state. If the

TABLE III: The interface providing by Isolayer to allocate resources.

Interfaces	Description & Return value	Syntax	Example
bool share()	Allocating channel resource to the object and return true if allocating successfully	bool share(IsolatedObj *object_name, float share†)	bandwidth(A,0.3)

control tables have some entries already, they will overwrite their tables and provide new isolation services.

C. Reduce the transmission overhead of control packets

Isolayer lowers the transmission overhead of the control packets from two aspects.

1) **Reduce the size of control packets.** The size of the whole MAC frame of the control packets that carry one object's information is about 146 bytes (36 bytes for the payload and about 110 bytes for overall header overhead [20]) on 802.15.4, larger than the Maximum Transmission Unit (MTU) in IEEE 802.15.4 that is 127 bytes. In other words, the control packet has be fragmented into two packets, which brings double transmission overhead. As shown in Fig. 4, the size of the `Src_addr` and `Dest_addr` fields is 32 bytes, about 88.9% of the total size. Therefore, Isolayer minimizes the size of these two fields to reduce the size of the control packets. For the control code that has no description of the source/destination address of the packets, Isolayer deletes these two fields. For the control code that only describes one of these two addresses, Isolayer reserves the corresponding address field and adds another bit "0" before the source address field if `Src_addr` is reserved to facilitate the identification of reserved address fields. If users have requirements for both the source and the destination address of the packets, these two fields will be reserved. In order to reduce the overall size, Isolayer will reuse the header compression module in 6LoWPAN [21] which is originally designed for IPv6 header compression. Besides, we add another byte that is 0 between consecutive "Value" and "Object ID" to distinguish the fields belong to the different objects,.
2) **Reduce the transmission times of control packets.** Broadcasting the control packets in the network is a feasible solution to deliver the isolation rules. However, it may introduce extra overhead (e.g., the control packets with the rules that 802.15.4 packets have 50% channel share are sent to BLE nodes). In order to eliminate such overhead, We have summarized a few rules (See Table IV) to guide the sending of control data packets. Unlike BLE and LoRa, 6LoWPAN enables multi-hop routing over 802.15.4. Therefore, the control packets must be transmitted to each 802.15.4 node in the network even if the source and destination address of the data packets have been specified. Generally, Isolayer reduces packet transmissions by 67% in theory by transmitting control packets strategically.

VI. Shared collision avoidance.

Wireless technologies already have some mechanisms to avoid collisions. For example, 802.15.4 relies on carrier sense (CSMA) to limit collisions and RTS-CTS to combat hidden terminals. BLE adopts a form of frequency-hopping spread spectrum (FHSS) called adaptive frequency hopping (AFH) to avoid packet collisions.

However, these mechanisms themselves are not enough in terms of channel isolation. First, they still let packets compete for channels in essence instead of reserving channel resources for the specific packets. Second, they may be invalid when cross-technology. For example, the RTS packets sent by the 802.15.4 node are incomprehensible to the BLE nodes. Therefore, Isolayer proposed that there should be a shared collision avoidance mechanism over heterogeneous wireless technologies in the isolation layer. Since LoRa uses license-free sub-gigahertz radio frequency bands like 433 MHz and is no overlap with BLE and 802.15.4 that operate on the 2.4GHz ISM band. We only consider 802.15.4 and BLE here.

We propose the avoidance mechanism on the top of cross-technology communication to solve the invalidness when cross-technology. XBee [22] is demonstrated as a ZigBee to BLE communication and BlueBee [19] is a BLE to ZigBee communication. Both of them can be implemented on commodity devices, requiring neither hardware nor firmware changes at the senders and the receivers.

Based on CTC, Isolayer borrows the idea of RTS/CTS mechanism and improves it to provide clear channels. Before transmission, the node sends a little packet containing a grant duration which will grants the channel around the transmitter to the transmitter. That is, upon overhearing such a packet, the surrounding nodes will be silent until the grant duration expires, and only the sender can transmit immediately. The grant duration d is related to the length L of the packet that will be transmitted and the data rate D of the wireless technology:

$$d \propto \frac{L}{D}. \tag{1}$$

VII. Conditionally-triggered isolation

For every packet in the activated node, Isolayer first judges whether it needs to be isolated. Isolayer extracts the protocol, packet type, source address and destination address of the packet to generate a packet ID. Then Isolayer compares the packet ID with each object ID in the control table. Once Equation 2 holds, the packet needs to be isolated.

$$P = P\&O \tag{2}$$

P is the ID of the packet while O is an object ID in the control table.

Next, the packet enters the corresponding packet queue according to the isolation rule it matches. The packet queue

TABLE IV: The relationship between the control code and the receiver of the control packets. "—" represents that the field can be empty or non-empty, "×" represents that the field must be empty, an exact string represents that field must be non-empty.

Conditions					Rules
Radio	Protocol	Packet Type	Source address	Destination address	Sending packets to #
R_802154	—	—	—	—	all 802.15.4 nodes
BLE	×	—	src_addr1	—	the BLE nodes whose address is src_addr1
LORA	×	—	src_addr2	—	the LoRa nodes whose address is src_addr2
×	—	—	src_addr3	×	the BLE/LoRa node whose address is src_addr3 if src_addr3 is an address of the BLE/ LoRa node **or** all 802.15.4 nodes if src_addr1 is an address of the 802.15.4 node
×	—	—	×	—	all IoT nodes

TABLE V: An example of the control table.

No.	Object ID	Share
1	0100000000..0110010	0010010
2	0000110010..0000000	1010110

module in the isolation layer performs the buffer management and packet scheduling. Isolayer borrows the basic idea of simple fair queueing [23] which is as known as *Weighted Fair Queueing* (*WFQ*) that provides output bandwidth sharing according to the preset weights. For example, suppose the weight of WFQ1 is 1, the weight of WFQ2 is 2 and the packet size in WFQ1 and WFQ2 are the same. Every time a packet from WFQ1 is sent, two packets from WFQ2 will be sent next. In other words, WFQ can allocate channel resources proportionally according to the weights of different queues. The key idea of WFQ is "virtual finish time". It calculates the virtual finish time for every packet in the queue by the following formula:

$$F_i = \max(A_i, Fi - 1) + \frac{L_i}{W}. \tag{3}$$

where i denotes a packet, F_i denotes the finish time, A_i denotes the arrival time of packet, F_{i-1} denotes the virtual finish time of last packet, L_i denote the packet length, W is the weight of the queue. Obviously, the virtual finish time is proportional to a packet's length in the traditional algorithm. However, since Isolayer reserves the channel and suppresses other transmitters, the isolation layer includes the silence durations of the packets in its calculations, as represented as:

$$F_i = \max(A_i, Fi - 1) + \frac{L_i + ST_i}{W}. \tag{4}$$

where ST_i is the silence time of packet i. Isolayer always reads the packet with the minimal virtual finish time from the queue and sends it firstly.

Providing WFQ-based Isolation services for packets with Isolayer requires two steps. First, create the packet queues according to the control table. Specifically, if the control table has N entries meaning there are $N + 1$ types of packets (N types that need to be isolated + another type that does not), Isolayer creates $N+1$ WFQs. Second, set the weights of these packet queues. The wights are set as equal to the share in the control table. As to the queue of the packets that does not need to be isolated, the weight is calculated as follows:

$$W = 100 - \sum_{i=1}^{N} W_i \tag{5}$$

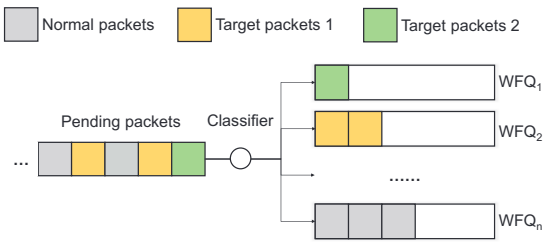


Fig. 5: The packet queues in the isolation layer. Packets enter the corresponding queues after the identification of the classifier.

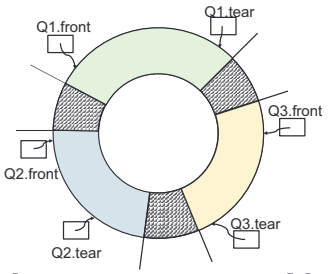


Fig. 6: The packet queue structure used by Isolayer. The shaded areas are the shared queue spaces.

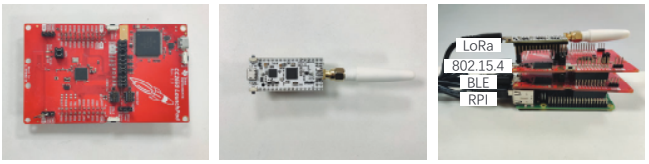
In summary, Isolayer provides two types of packet queues: a FIFO queue when nodes are unactivated and several WFQs when activated, as shown in Fig. 5. When the nodes are unactivated, all packets enter the FIFO queue and are scheduled by the time they arrive. Once the nodes are activated, Isolayer creates WFQs. Pending packets enter the corresponding queue after the identification of the classifier. However, the number of entries in the control table is variable. When there are dozens of entries in the control table, generating the packet queues may occupy too much dynamic memory, harming the implementation of Isolayer on low-end IoT nodes.

Isolayer uses the idea of circular queue [24] to solve the problem, as shown in Fig. 6. Isolayer preallocates the memory to the circle queue shared by multiple packet queues. Besides, Isolayer reserves some space which can expand the queues on its both end to achieve flexible allocation.

VIII. Evaluation

A. Implementation

Hardware and OS. We implement Isolayer on CC2650 (ROM 128kB, RAM 20kB, multi-standard radio chip support-



(a) CC2650 Launchpad (b) Heltec LoRa Node 151 (c) Gateway

Fig. 7: IoT nodes for the implementation.

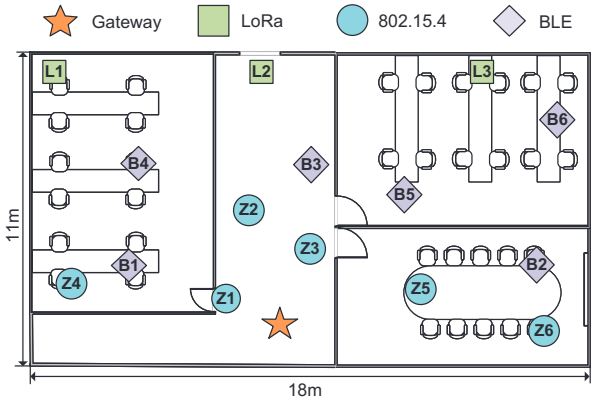
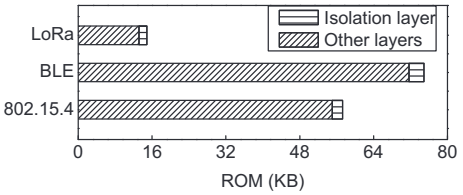


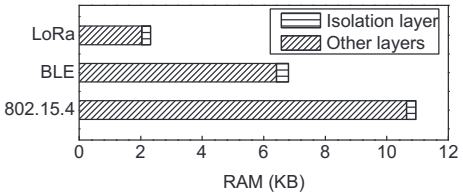
Fig. 8: An indoor testbed for Isolayer.

TABLE VI: Default parameters setting of Isolayer.

Parameters	Value
The number of entries in control table	5
The capacity of packet queue	48
The number of the radios supported	1



(a) ROM size



(b) RAM size

Fig. 9: ROM and RAM usage of Isolayer.

ing 802.15.4/ZigBee and BLE) [3] and Heltec LoRa node 151 (ROM256kB, RAM32kB). [25]. We also implement Isolayer on a gateway comprising of a Raspberry Pi 3 (RPI) [26] connected with one Heltec LoRa node 151 and two CC2650 chips. The isolation layer and the above run on the RPI, providing unified isolation services, while the MAC and the lower layer run on the radio devices. Fig. 7 shows these three types of IoT nodes.

We implement Isolayer on RIOT [8], an open-source operating system that supports a rich set of protocols and reuse the GNRC protocol stack [27] and BLE NimBLE stack [28] of RIOT.

Setup. In order to facilitate the evaluation, we use a laptop to act as the cloud/edge to parse the control code and generate the control packages. It connects to the gateway by a network cable. We mainly use a mixed indoor testbed for the evaluation (see Fig. 8). Six 802.15.4 nodes, six BLE nodes and three LoRa nodes are deployed indoor. We configure the transmission power of 802.15.4 nodes to form a two-hop topology. The BLE and LoRa nodes communicate directly with the gateway. When not specified, the payload length of data packets is 20 bytes. The LoRa SF is set to 7, the bandwidth is set to 125kHz and the CR is set to 4/8. The connection interval of BLE is set to three seconds.

We perform a systematic evaluation on Isolayer. We compare Isolayer with two network stacks, i.e., FWP [1] providing isolation between different protocols on 802.15.4 network and original GNRC implemented in RIOT OS [27].

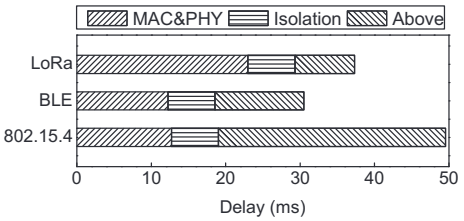
B. Overhead of Isolayer

1) ROM and RAM: The lightweight implementation of Isolayer is essential because it will be installed on resource-constrained IoT nodes as well. We quantify the ROM and RAM of Isolayer with its default implementation on RIOT OS. Some parameters that may influence memory usage are set as Table VI.

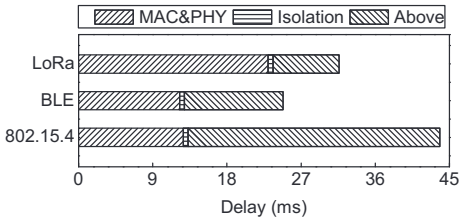
Figure 9 shows the ROM and RAM of Isolayer on different radios. Compared with the original network stacks, Isolayer's ROM and RAM increase about 4.0% and 2.9% on 802.15.4, 4.5% and 6.1% on BLE, 12.6% and 13.6% on LoRa. Though larger memory usage, the implementation of Isolayer is still lightweight for many existing IoT nodes. For example, the Imote2 node has 256KB RAM and 32MB ROM; Arduino ZERO has 32KB RAM and 256KB ROM; TI CC2538 has 32KB RAM and 512KB ROM.

2) Delay: We evaluate the overhead of Isolayer in terms of communication delay with a light traffic load. The transmitter A sends a 20-byte packet to the gateway per five seconds. Once receives a packet from node A, the gateway relays it to another node B. We record the end-to-end delays of 200 packets when the isolation layer is activated and not activated, respectively. When activated, the isolation layer allocates the packets from A to B with 100% channel share.

Figure 10 shows the average end-to-end delay of the 802.15.4, BLE and LoRa packets. Since 802.15.4 node is implemented with a full TCP/IP stack compared with LoRa and BLE, its delay is the maximum. Compared with GNRC, activated Isolayer costs 11.29%, 25.96% and 20.29% more time on 802.15.4, BLE and LoRa, respectively while the unactivated Isolayer costs 1.39%, 2.43% and 1.94% more time. When unactivated, Isolayer only performs one more step to confirm its state by checking whether the control table is empty before sending a packet. By introducing the unactivated state, the negative impact of the isolation layer can be reduced significantly when the isolation service is not needed.



(a) Delay performance when the isolation layer is **activated.**



(b) Delay performance when the isolation layer is **unactivated.**

Fig. 10: Delay performance on different wireless technologies.

TABLE VII: Time of taking effect on different wireless technologies.

	802.15.4	BLE	LoRa
Time	68.3 ms	52.1 ms	132.6 ms

C. Time of taking effect

We study the time of Isolayer taking effect in this section. The topology is that a gateway connects to an IoT node (i.e., 802.15.4 node, BLE node or LoRa node). When receiving the control packets, the gateway will forward them to the IoT node. We record the time interval from the gateway relaying the control packet to the node starting to provide isolation services. There are no other packets that need to be transmitted by the gateway. The control packets' payload sizes of different wireless technologies are the same.

The results are shown as Table VII which are averaging from 50 events. The LoRa node has the maximum time interval. The reasons are two-fold: 1) LoRa has the lowest data rate. In the settings of SF=7 and bandwidth = 125kHz, the data rate is about 5.47 kbps, much smaller than the 802.15.4's and BLE's. 2) LoRa has the maximum rendezvous delay. As specified in the LoRaWAN specification [29], the gateway has to wait for the uplink packets to perform the downlink transmissions.

D. Compatibility with heterogeneous network

In this section, we evaluate Isolayer in terms of its collision avoidance mechanism in a heterogeneous network. One 802.15.4 node sends 20-byte UDP packets to the gateway on channel 14. Four BLE nodes send 20-byte packets to the gateway as well. In order to evaluate the avoidance mechanism of Isolayer, we disable the adaptive channel-hopping mechanism of the BLE nodes and make them send packets on channel 8 that overlaps channel 14 of 802.15.4. We make the traffic load increase by setting the packets sending interval of BLE and

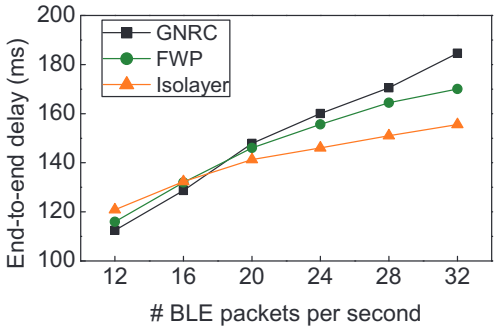


Fig. 11: The end-to-end delay of 802.15.4 packets when coexisting with BLE packets.

study the end-to-end delay of 802.15.4 packets as the traffic load varies. In this case, Isolayer allocates 50% of the channel resources to 802.15.4 packets.

Figure 11 shows the results. When the traffic load increases, Isolayer achieves a 15.8% reduction compared with GNRC and an 8.8% reduction compared with FWP. Since the 802.15.4 node implemented with GNRC lacks a collision avoidance mechanism, it backs off repeatedly if the channel is jam-packed. FWP uses GTS to provide collision avoidance, but it is invalid to the heterogeneous nodes. The 802.15.4 node implemented with FWP still suffers from the MAC layer's back-off mechanism because of the transmission of BLE packets. The inefficiency of the collision avoidance mechanism and the delay introduced by the isolation layer cause FWP's maximum end-to-end delay. Isolayer provides a shared avoidance mechanism on heterogeneous wireless technologies and thus has a better performance.

E. Case study

In this section, we evaluate Isolayer from two cases mentioned in Section IV.

1) DDos attack: In this case, we try to prove the ability of Isolayer in resisting denial-of-service (DDoS) attacks. In this case, four 802.15.4 node act as the attackers that send in total of 20 20-byte UDP packets to the gateway per second continuously. A BLE node acts as the victim and sends one 20-byte packet to the gateway per second. The Isolayer allocates the packets from the BLE node 20% channel share. We record the BLE packets' channel share on the gateway before and after the isolation layer's activation, as shown in Fig. 12.

Obviously, the channel share of BLE packets goes to stabilization (i.e., 5%) after the gateway starts to provide isolation service at 30 sec. The primary reason is that Isolayer reserves an empty packet queue for BLE packets. Without isolation, the BLE packets have to share the packet queue with other packets. The fake packets from the attackers occupy the most space of queue in the jam-packed gateway. Thus the BLE packets may be dropped. With isolation, the BLE packets have a single packet queue to avoid being dropped. Besides, the scheduling mechanism of Isolayer guarantees their channel share, which means they will not wait long before being sent.

2) Industrial IoT: We use a simulation case of an industrial IoT (IIoT) application, which has been introduced in

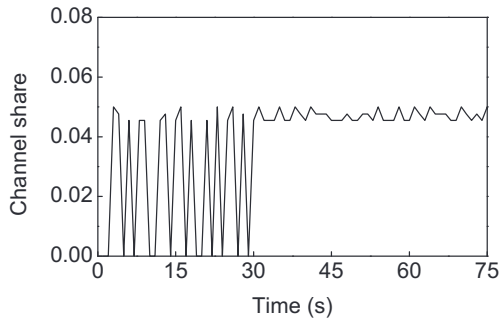


Fig. 12: Channel shares of the BLE packets in a simulated DDoS attack.

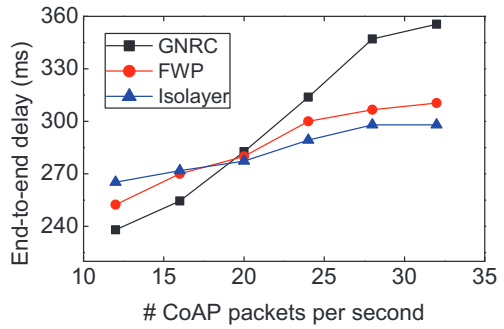


Fig. 13: The end-to-end delay of the MQTT packets as the traffic load increases.

Section IV to prove the role of Isolayer in wireless IIoT. In this case, four CC2650 nodes upload CoAP packets to the cloud (a laptop here) with 802.15.4 through the gateway. An MQTT broker runs on the laptop. A CC2650 acts as an MQTT subscriber while another CC2650 acts as the publisher publishing the message to the broker every five seconds. We record the delay from the publisher sending the message to the subscriber receiving the message. Isolayer provides the MQTT packets with 50% channel share and the isolation layer is always activated. As the traffic load increase, the results are displayed in Fig. 13.

When the traffic load is light (e.g., BLE nodes send three As the traffic load increase, the end-to-end delay of the three stacks all increases. The reasons are two-fold. First, the probability of packet collisions increases, resulting in retransmission. Second, packets have to wait more time in the packets queue and may even be dropped.packets per second), Isolayer has the maximum delay because of the overhead of the isolation layer. However, the introduced delay can be eliminated by close the isolation service that is unnecessary when the traffic load is light. When the traffic load is relatively heavy, Isolayer has a better performance achieving an 18.6% reduction compared with GNRC and a 3.9% reduction compared with FWP. The difference comes from that Isolayer provides an individual packet queue and a certain share of the clear channel for MQTT packets.

IX. Related Work

Isolayer draws insights from many existing works. We divide them into three categories: packet scheduling algorithms, interference detection and network isolation.

Packets scheduling algorithms. In the last few decades years, many classical packet scheduling algorithms [23], [30]–[32] which are capable of providing guaranteed QoS have been developed. However, most of them are designed, ignoring the extra delay introduced by retransmission or back-off, making them inapplicable to the wireless network due to the too unreliable wireless channels. Therefore, the problem of providing QoS over unreliable wireless channels has received growing interest in recent years. Jaramillo et al. [33] propose a framework for developing scheduling policies in ad hoc networks with real-time flows and provides online scheduling policies for a special case. Hou [34] proposes a general approach for designing scheduling mechanisms for real-time traffic over time-varying wireless channels. Yang et al. [35] study deadline-aware scheduling with adaptive network coding (NC) for real-time traffic over a single-hop wireless network.

Compared with previous works focusing on a single radio, our work is an architecture applicable to heterogeneous networks, which provides multi-level isolation considering collision among different wireless technologies.

Collision avoidance. SoNIC [36] enables resource-limited sensor nodes to detect different interference sources (e.g., WiFi and Bluetooth) by extracting the distinct patterns in 802.15.4 packets, e.g., variances of in-packet RSSI series, link quality indication, etc. DOF [37] utilizes the autocorrelation of signals to sensing the spectrum occupancies of different types of signals. Smoggy-Link [38] maintains a link model to describe and trace the relationship between interference and link quality of the sender's outbound links. With such a link model, Smoggy-Link can obtain fine-grained spatiotemporal link information to perform adaptive link selection and transmission scheduling. ZiSense [39] is a low-duty cycling mechanism resilient to interference. It detects ZigBee signals by short-term features extracted from the time-domain RSSI sequence and wakes up nodes accordingly.

Most of the above works focus on interference detection of the specific wireless technology (e.g., 802.15.4 in SoNIC [36] and ZiSense [39]) and hardly maintain the effectiveness when detecting interference of other wireless technologies. Isolayer extends an existing work and provides a shared collision avoidance mechanism for different wireless technologies (e.g., 802.15.4, WiFi and LoRa).

Network isolation. The motivation for network isolation is non-interference between different protocols/applications. The idea that the network should support concurrent operation for multiple protocols/applications is not new in the Internet. TCP slows down its data generation when it encounters a packet loss, which is one of the keys to Internet scalability. This congestion control feature exists at layer 4 because the narrow waist of the Internet is layer 3. However, the narrow waist has moved to the bottom layer in the IoT network due to the scarce channel resources. Choi et al. [1] introduce a new isolation layer in sensor network communication stacks. It achieves isolation and fairness by shared collision avoidance and fair queuing across protocols in sensor networks. CerborOS [40]

is a resource-secure OS for sharing IoT devices. It uses a code interpreter for instruction-level monitoring and managing running apps and resource contracts to define applications resource usage.

These work have their contributions. However, they ignore the varying requirements in practice and only provide single-dimensional isolation. Even though, the above existing works shed light on an important aspect in designing Isolayer, that is how to provide reliable isolation between different flows. Isolayer is an extension of these works, which is dedicated to providing user-customizable isolation solutions in heterogeneous IoT networks. Besides, these works also indicate the direction for future work, such as designing an architecture that can isolate the computation and storage resources for different objects.

X. Conclusion

In this paper, we argue that IoT network stacks should have an isolation layer. Based on the above idea, we design **Isolayer**, an isolation layer design providing *flexible* and *fine-grained* channel isolation services in *heterogeneous* IoT networks. Isolayer provides four levels of isolation services (i.e., protocol level, packet-type level and source-/destination-address level) according to the users' requirements and still provides effective isolation services between packets with different wireless technologies. We have implement Isolayer in the existing IoT network stack in RIOT OS and perform extensive evaluations on the realistic IoT nodes. Results have proved that Isolayer is lightweight and provides effective isolation services in various scenarios. Isolayer is concerned with the networks' heterogeneity trends and laid the foundation for the promotion of multi-radio applications to some extent.

References

[1] J. I. Choi, M. Kazandjieva, M. Jain, and P. Levis, "The Case for a Network Protocol Isolation Layer," in *Proceedings of ACM SenSys*, 2009.

[2] "LoRa," https://www.semtech.com/lora.

[3] "TI CC2650 SimpleLink™ Multistandard Wireless MCU," https://www.raspberrypi.org/.

[4] M. B. Attia, K. Nguyen, and M. Cheriet, "Dynamic QoE/QoS-Aware Queuing for Heterogeneous Traffic in Smart Home," *IEEE Access*, vol. 7, pp. 58 990–59 001, 2019.

[5] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[6] J. Pan, R. Jain, S. Paul, T. Vu, A. Saifullah, and M. Sha, "An Internet of Things Framework for Smart Energy in Buildings: Designs, Prototype, and Experiments," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 527–537, 2015.

[7] B. L. Risteska Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, pp. 1454 – 1464, 2017.

[8] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proceedings of IEEE INFOCOM WKSHPS*, 2013.

[9] "FDA confirms that St. Jude's cardiac devices can be hacked," https://money.cnn.com/2017/01/09/technology/fda-st-jude-cardiac-hack/.

[10] "Threat Advisory: Mirai Botnet," https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/akamai-mirai-botnet-threat-advisory.pdf.

[11] "Casino Gets Hacked Through Its Internet-Connected Fish Tank Thermometer," https://money.cnn.com/2017/01/09/technology/fda-st-jude-cardiac-hack/.

[12] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[13] A. Munshi, N. A. Alqarni, and N. Abdullah Almalki, "DDOS Attack on IoT Devices," in *Proceedings of IEEE ICCAIS*, 2020.

[14] K. Sonar and H. Upadhyay, "A survey: DDOS attack on Internet of Things," *International Journal of Engineering Research and Development*, vol. 10, no. 11, pp. 58–63, 2014.

[15] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "RFC 7252: The constrained application protocol (CoAP)," *Internet Engineering Task Force*, 2014.

[16] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *Proceedings of IEEE ISSE*, 2017, pp. 1–7.

[17] "Message Queuing Telemetry Transport," http://mqtt.org/.

[18] S. Kim, H. Choi, and W. Rhee, "IoT home gateway for auto-configuration and management of MQTT devices," in *Proceedings of IEEE ICWiSe*, 2015, pp. 12–17.

[19] W. Jiang, Z. Yin, R. Liu, Z. Li, S. M. Kim, and T. He, "Bluebee: a 10,000 x Faster Cross-technology Communication via PHY Emulation," in *Proceedings of ACM SenSys*, 2017.

[20] S. Kumar, M. P. Andersen, H.-S. Kim, and D. E. Culler, "Performant TCP for Low-Power Wireless Networks," in *Proceeding of USENIX NSDI*, 2020.

[21] G. Mulligan, "The 6LoWPAN architecture," in *Proceedings of the IEEE EmNets*, 2007, pp. 78–82.

[22] W. Jiang, S. M. Kim, Z. Li, and T. He, "Achieving receiver-side cross-technology communication with cross-decoding," in *Proceedings of ACM MobiCom*, 2018.

[23] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Proceedings of ACM SIGCOMM*, vol. 19, no. 4, pp. 1–12, 1989.

[24] P. Basu Ray, "Data structures: Circular queue," 2021.

[25] "LoRa Node 151," https://heltec.org/project/lora-node-151/s.

[26] Raspberry Pi, https://www.raspberrypi.org/, 2020.

[27] Generic (GNRC) network stack, https://riot-os.org/api/group_net_gnrc.html, 2020.

[28] Apache NimBLE, https://github.com/apache/mynewt-nimble, 2021.

[29] LoRa Alliance Technical Committee., "LoRaWAN 1.1 specification. Standard V1," 2017.

[30] J. C. R. Bennett and H. Zhang, "WF2Q: Worst-Case Fair Weighted Fair Queueing," in *Proceedings of IEEE INFOCOM*, 1996.

[31] F. Checconi, L. Rizzo, and P. Valente, "QFQ: Efficient Packet Scheduling With Tight Guarantees," *IEEE/ACM Transactions on Networking*, vol. 21, no. 3, pp. 802–816, 2013.

[32] C. Guo, "G-3: An O (1) time complexity packet scheduler that provides bounded end-to-end delay," in *Proceedings of IEEE INFOCOM*. IEEE, 2007.

[33] R. Li and A. Eryilmaz, "Scheduling for End-to-End Deadline-Constrained Traffic With Reliability Requirements in Multihop Networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1649–1662, 2012.

[34] I. Hou, "Scheduling Heterogeneous Real-Time Traffic Over Fading Wireless Channels," *IEEE/ACM Transactions on Networking*, vol. 22, no. 5, pp. 1631–1644, 2014.

[35] L. Yang, Y. E. Sagduyu, J. Zhang, and J. H. Li, "Deadline-aware scheduling with adaptive network coding for real-time traffic," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1430–1443, 2015.

[36] F. Hermans, O. Rensfelt, T. Voigt, E. Ngai, L. Nordén, and P. Gunningberg, "SoNIC: Classifying interference in 802.15.4 sensor networks," in *Proceedings of ACM/IEEE IPSN*, 2013, pp. 55–66.

[37] Hong, Steven Siying and Katti, Sachin Rajsekhar, "DOF: A Local Wireless Information Plane," 2011.

[38] Meng Jin, Y. He, X. Zheng, Dingyi Fang, Dan Xu, Tianzhang Xing, and Xiaojiang Chen, "Smoggy-Link: Fingerprinting interference for predictable wireless concurrency," in *Proceedings of IEEE ICNP*, 2016.

[39] X. Zheng and Z. Cao and J. Wang and Y. He and Y. Liu, "Interference Resilient Duty Cycling for Sensor Networks Under Co-Existing Environments," *IEEE Transactions on Communications*, vol. 65, no. 7, pp. 2971–2984, 2017.

[40] S. Akkermans, W. Daniels, G. S. Ramachandran, B. Crispo, and D. Hughes, "CerberOS: A Resource-Secure OS for Sharing IoT Devices." in *Proceedings of EWSN*, 2017.