TinyNet: a Lightweight, Modular, and Unified Network Architecture for the Internet of Things

Wei Dong[1,2], Jiamei Lv[1,2], Gonglong Chen[1], Yihui Wang[1]
Huikang Li[1], Yi Gao[1,2], Dinesh Bharadia[3]
[1]College of Computer Science, Zhejiang University.
[2]Alibaba-Zhejiang University Joint Institute of Frontier Technologies.
[3]University of California, San Diego
{dongw,lvjm,chengl,wangyh,lihk,gaoyi}@zju.edu.cn,dineshb@eng.ucsd.edu

Abstract

Interoperability among a vast number of *heterogeneous* IoT nodes is a key issue. However, the communication among IoT nodes does not fully interoperate to date. The underlying reason is the lack of a lightweight and unified network architecture for IoT nodes having *different* radio technologies. In this paper, we design and implement TinyNet, a *lightweight*, *modular*, and *unified* network architecture for representative low-power radio technologies including 802.15.4, BLE, and LoRa. The *modular* architecture of TinyNet allows us to simplify the creation of new protocols by selecting specific modules in TinyNet. We implement TinyNet on realistic IoT nodes including TI CC2650 and Heltec IoT LoRa nodes. We perform extensive evaluations. Results show that TinyNet (1) allows interoperability at or above the network layer; (2) allows code reuse for multi-protocol co-existence and simplifies new protocols design by module composition; (3) has a small code size and memory footprint.

CCS Concepts

• **Networks** → **Network design principles**; **Network protocol design**.

Keywords

Network architecture, Internet of Things, Interoperability.

ACM Reference Format:
Wei Dong[1,2], Jiamei Lv[1,2], Gonglong Chen[1], Yihui Wang[1], Huikang Li[1], Yi Gao[1,2], Dinesh Bharadia[3]. 2022. TinyNet: a Lightweight, Modular, and Unified Network Architecture for the Internet of Things. In *The 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22), June 25–July 1, 2022, Portland, OR, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3498361.3538919

1 Introduction

In the last few years, the Internet of Things (IoT) has become one of the most promising and exciting developments in technology and business. Low-power wireless radio technologies form the basis of many IoT applications and are continuously evolving over the years. For example, 802.15.4 is a relatively mature technology and has attracted a lot of research attention from academia in the past years [30, 43]. Bluetooth and BLE (Bluetooth Low Energy) are widely used in our daily life. LoRa is a representative LPWAN (Low Power Wide Area Network) technology and gains increasing research interests [14, 42]. These low-power wireless radio technologies have drastically different implementations for their protocol stacks (see Figure 1(a)).

Most IoT devices do not directly support IP and must rely on the gateway to provide IP support, which in turn provides interoperability to the rest of the Internet. In other words, most IoT devices are not *directly interoperatable* with the rest of the Internet or other IoT devices. For example, IoT nodes without IP (due to strict resource constraints) cannot use application-layer IoT protocols, e.g., MQTT [40], to specify high-level semantics to communicate with each other. The so called "missing interoperability" could lead to *substantial* threat to the predicted economic value, as pointed out by McKinsey analysis [36]. Lack of interoperability means that service providers are bound to the IoT device or software offered by a single provider and must stick with it. This may bring the potential risk of higher operating costs later on, as well as product functionality and stability issues [35].

To overcome this problem, the research community has proposed several proposals. For example, 6LoWPAN [24] was proposed a decade ago. This standard encourages the use of IPv6 and specifies how to format IPv6 packets in a *compact* way over low-power wireless links. Recent proposals have provided low-cost implementation of IP for different radio stacks. For example, OpenThread [28] offers IPv6-over-802.15.4 stack to problems such as interoperability, security, power, and architecture requirements. BLEach [47] designs an IPv6-over-BLE stack which allows to fine-tune communication performance. TCPlp [30] designs a full TCP/IP stack over 802.15.4 networks so that 802.15.4 nodes can be connected with existing TCP/IP networks as part of the Internet of Things. The above research works are essential building blocks of the proposal of *Web of Things* which is at the center of a standardization effort at W3C [54].

Existing IoT network stacks, e.g., stacks in LiteOS [23] and TencentOS Tiny [49], try to be Internet compatible. However, they fall short in providing a unified and modular architectural approach (see Figure 1(b)). More specifically, these stacks only provide unification above the network layer for BLE and 802.15.4, while the lower layers, as well as the LoRa stack, are still non-interoperatable. The above difficulties come from the fact that

MobiSys '22, June 25–July 1, 2022, Portland, OR, USA

ACM ISBN 978-1-4503-9185-6/22/06…$15.00
https://doi.org/10.1145/3498361.3538919

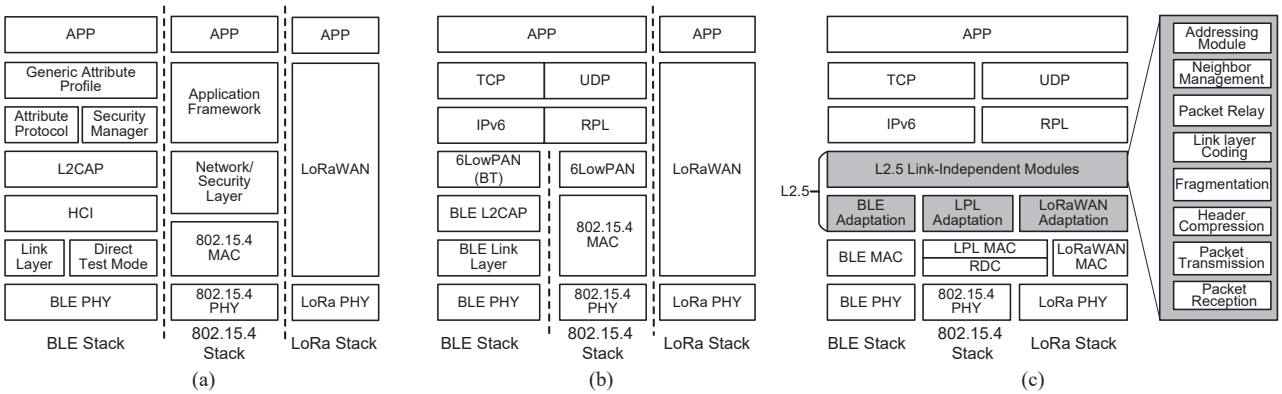


Figure 1: Three different IoT network architectures. (a) Vertically-integrated architecture (b) Existing Internet compatible architecture (e.g. stacks in Huawei LiteOS and TencentOS Tiny) (c) TinyNet.

different radio technologies often require drastically different implementations, causing difficulties in interoperating among IoT nodes with different radio technologies.

In this paper, we present TinyNet, a lightweight network architecture and its implementation that provide IP support to multiple IoT radio technologies. TinyNet has the following design goals:

- **Unified**: TinyNet should be able to unify many different radio technologies at the network layer so that heterogeneous IoT nodes can seamlessly communicate with each other at or above the network layer, using high layer protocols.
- **Modular**: The design should be modular so that new protocols can be easily adopt the framework and extend it to any new radio technologies.
- **Lightweight**: The implementation should be lightweight, allowing the installation on low-end IoT nodes. Furthermore, on gateways, it should also be lightweight to support multiple radios without repeating each protocol in its entirety.

It is challenging to achieve unification of multiple radio technologies with proper modularity. We introduce layer 2.5, i.e., L2.5, to allow multiple link technologies and multiple network-layer routing protocols to coexist and evolve independently of each other (see Figure 1(c)). Within L2.5, we introduce two MAC-relevant layers and three abstractions (i.e., property, service and event) to decouple the interfaces to upper modules and the specific implementations in other network stacks. We have also proposed a set of novel techniques, including *unified neighbor table design* (Section 4.3), *automatic synchronization* of transmission/reception slots (Section 4.4), *conflict graph based radio scheduling* for multi-radio platforms (Section 4.2), to deal with specific challenges in handling heterogeneous radios, which has not been addressed in existing literature [43, 47].

We implement TinyNet on CC2650 (ROM 128kB, RAM 20kB, multi-standard radio chip supporting 802.15.4/ZigBee and BLE) and Heltec IoT LoRa node 151 (ROM 256kB, RAM 32kB). We evaluate its performance extensively (Section 7). Results show that:

- TinyNet allows interoperability among heterogeneous radio technologies, including 802.15.4, BLE, LoRa and WiFi, at or above the network layer (Section 7.1). We also illustrate that supporting TCP/IP is feasible for many commonly used embedded systems. For example, our results show that (1) for the implementation of a single radio, TinyNet consumes 6.146–6.588 KB in RAM and 48.013–51.630 KB in ROM; (2) for the implementation of three radios, it consumes up to 10.367 KB in RAM and 77.411 KB in ROM. This fits in many widely used low-end embedded systems such as Arduino ZERO (ROM 256KB, RAM 32KB), STM32F103 (ROM 512KB, RAM 64KB), CC2650 (ROM 128KB, RAM 20KB), etc.
- TinyNet simplifies new protocol design by module composition. Two newly proposed protocols (Rateless BLE and RPL over LoRaWAN) can be easily composed by selecting existing modules in TinyNet (Section 5).
- In terms of code size and memory footprint, TinyNet achieves significant reduction compared with two existing network stacks—Tencent stack (stack implementation in TencentOS Tiny [49]) and GNRC (stack implementation in RIOT [15]). Specifically, TinyNet occupies 39.9% less code size, and 11.2% less memory footprint compared with Tencent stack, and 45.5% less code size and 9.9% less memory footprint compared with GNRC (Section 7.3).
- TinyNet achieves better or comparable performance in terms of communication delay and energy efficiency compared with Tencent stack and GNRC. (Section 7.4 and Section 7.5).

We believe TinyNet would be an excellent experimental platform that will allow the research community to experiment broadly and deeply in order to investigate important questions that arise when scaling up IoT networks as well as rapidly developing new protocols from the ground up. For example, in the smart home applications, by enabling RPL and multi-hop over BLE, hundreds of BLE nodes in different rooms can be connected to the Internet via a single BLE gateway. Researchers can also seek for a variety of opportunities to optimize the end-to-end network performance.

2 Motivation

We study several real-world scenarios and use cases to illustrate the need for a lightweight, modular and unified network architecture for IoT.

Motivation for unification (device control and remote monitoring). Many IoT devices have already been deployed in homes. People can remotely control the IoT devices, e.g., turn

on a smart LED, through a specific app that interacts with the manufacturer's cloud. Devices without IP support are hidden behind the cloud and cannot be accessed without the translation and relaying of cloud. This has encouraged IoT applications to develop as *vertically-integrated* silos, where devices cooperate only within an individual application or a particular manufacturer's ecosystem. As such, people cannot control the devices without installing specific apps. Moreover, it introduces difficulties for cross-company device communication and prevents many interesting applications requiring multi-device interactions. On the other hand, allowing devices to seamlessly communicate with each other, i.e. machine to machine (M2M) communication, can have a wide range of application scenarios. For example, in the remote monitoring scenario, a screen display can directly connect to an IP camera for retrieving the video stream.

If devices are equipped with a unified stack and can fully interoperate, we can embrace the fast-growing Web technologies and can access the devices through a unified interface, e.g., RESTful APIs. An appealing scenario we can envision is the semantic web of things described in [21]. In this scenario, each IoT device can self-describe its capabilities in standardized ways. Applications can then search for IoT devices (e.g., in their local area) and access/control them via the Web. We will eventually enter the era of the Web of Things.

Motivation for lightweight implementation (memory shortage). Many IoT devices are equipped with multiple radios, e.g. one energy-efficient radio for service discovery and one high-throughput radio for data transfer [4]. [56] gives a detailed description of 13 use cases across regions combining WiFi and LoRaWAN. Another recent trend is the increasing use of multi-standard radio chips, which are rapidly penetrating the IoT device market. For example, the CC2650 chip supports two radio standards, i.e., BLE and 802.15.4, through a single radio interface. Multi-standard chips can also simultaneously offer multiple communication interfaces by carefully consolidating multiple radio stacks [26]. However, supporting multiple radio stacks requires a large amount of memory. For example, it requires over 128kB of program memory by simultaneously supporting 802.15.4 and BLE with RIOT GNRC (see Section 7.3). It prevents its use on resource-constrained devices such as CC2650 and CC1350 (both with 128kB program memory).

If we carefully optimize and modularize the network stacks for different radios, common modules can be reused by many radio stacks, thereby reducing the total memory consumption.

Motivation for modularity (protocol innovation). Many IoT systems are deployed in application-specific scenarios and have a heavy need for optimization or re-implementation of some functionalities. For example, a BLE network with long-distance communication may require adding link-layer coding (e.g. Rateless coding) and multihop transmission (e.g. RPL) functionalities for improving the network performance. Unfortunately, existing implementations are bound to specific radio technologies, e.g. there are no available implementations for BLE, although there exist rateless coding and RPL implementations for ZigBee [55]. The creation of new protocols thus requires more effort to reorganize functionalities or even reimplement them from scratch. The lack of modularity also causes the porting of applications onto different protocols to become non-trivial.

If there exists a carefully designed modular network architecture for heterogeneous radio technologies, the creation of new protocols will be greatly simplified.

Motivation for building universal gateways. For an IoT device to connect to the Internet, a gateway is essential. Today, many IoT devices require application-layer gateways. Therefore, people need to buy different gateways for different IoT devices, with application-specific protocols installed in the gateway. In contrast, other networks are seamlessly interoperable with the rest of the Internet. For example, accessing a new web application from a laptop does not require any new functionality at the Wi-Fi access point.

If there exists a lightweight, unified and Internet-compatible network stack installed at each IoT device, the gateway design would be greatly simplified by avoiding application-specific considerations. Ultimately, we can create universal multi-radio gateways supporting a variety of IoT devices from different vendors.

3 TinyNet Architecture

Figure 1(c) shows the network architecture of TinyNet where the boxes indicate different modules. The current implementation of TinyNet unifies three radio technologies, including 802.15.4, BLE, and LoRa. On top of PHY layers, there are dedicated MAC protocols, e.g., BLE MAC [20], LoRaWAN [8] and LPL (Low Power Listening) MAC [11]. It is worth noting that TinyNet also allows LoRa to reuse the MAC protocols originally designed for 802.15.4 PHY, i.e., RDC (Radio Duty Cycle Control) and LPL MAC. An important part of TinyNet is located at L2.5, i.e., an abstraction layer between the link and network layer.

Why exists L2.5. The goal of introducing a layer 2.5 in TinyNet is to provide a unified interface to a wide range of data link and physical layer technologies that allows network layer and the above protocols to operate efficiently through link independent optimizations. Positioning at L2.5, we allow multiple link technologies (e.g., LoRa, 802.15.4, Z-wave, etc) and multiple network protocols (e.g., RPL [57], WirelessHART [46], BLE mesh [39], etc) to coexist and evolve independently of each other in the same way the IP layer allows transport protocols and link layer technologies to evolve independently in today's Internet.

Modules inside L2.5. In designing L2.5, we first start from SP [43], an existing L2.5 layer to include the neighbor management and packet queue (i.e., message pool in SP). While these two modules are useful for providing common services to the network layer. They are, however, insufficient since they are lacking support for key services TinyNet wants to provide, i.e., (1) support for IPv6 [25] and 6LoWPAN [24] which forms the basis for network interoperability; (2) support for multihop communication which is valuable for increasing the wireless communication range; (3) reliability.

Table 1 summarizes the relationship between the desired services and the L2.5 modules that provide them. To further increase code reuse, we abstract two additional modules, packet transmission and packet reception to deal with different packet formats, fragmentation and coding mechanisms. Some modules,

Table 1: Services provided by, and modules implemented in TinyNet.

Services	Modules
Common network services	Neighbor management, Packet queue
Support for different link technologies	BLE adaptation, LoRaWAN adaptation, LPL adaptation
Support for 6LoWPAN	Header compression, Fragmentation
Multihop support	Packet relay, Addressing module
Reliability	Link layer coding

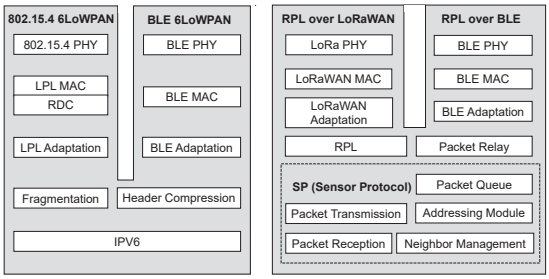


Figure 2: Basic decomposition of four protocols. Common functions can be shared, reducing the total number of components in the system.

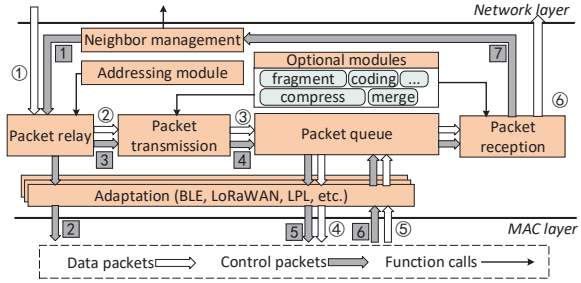


Figure 3: Key procedures for data and control packets.

e.g., fragmentation and header compression, are necessary for supporting 6LoWPAN and IPv6. Some other modules, e.g., BLE adaptation and LoRaWAN adaptation, are necessary for unifying different radio technologies. By carefully abstracting common functionalities, many modules of TinyNet can be reused in various protocols. For example, the fragmentation and the header compression module can be used for both BLE 6LoWPAN and 802.15.4 6LoWPAN (see Figure 2).

Compared with earlier efforts [13, 29, 31, 43], TinyNet clearly goes a step further—we have shown in Section 5 that earlier protocols, e.g., SP [43], RPL-over-BLE [31] can be easily created by selecting specific modules in TinyNet. More importantly, TinyNet can compose new protocols, e.g., RPL-over-LoRa and rateless BLE, which cannot be easily created by all prior architectures. Another key difference is that while all existing architectures focus on one specific radio technology, i.e., 802.15.4 [13, 29, 43] or BLE [31], TinyNet intends to design an L2.5 that works on many different radio technologies. Achieving the abovementioned salient features requires more systematic considerations in designing many TinyNet modules (Section 4).

Data and control packet procedures. Figure 3 shows the procedure of forwarding data and control packets from the network layer. In data plane, a packet first goes through the packet relay module which allocates the transmission slots. Then the packet transmission module interacts with the packet queue module to schedules packet transmission. Finally, the MAC adaption module sends the data packet on the radio interface. In control plane, most of the steps are similar excluding several differences: at step ②, the packet relay module relies on the abstracted APIs of the adaptation layer to synchronize the slots for advertising and scanning.

4 Module design

We present the design details of some key modules in TinyNet. Figure 1(c) shows a list of modules we have implemented in TinyNet, including BLE/LPL/LoRaWAN adaptation modules, addressing module, neighbor management module, packet relay, etc. The full descriptions of these modules can be found in our technical report [48].

4.1 MAC and MAC-adaptation modules

The adaption module is the key for TinyNet to incorporate many radio technologies. TinyNet introduces two layers to achieve this unification.

First, TinyNet's MAC presentation layer intends to describe each MAC's full functionalities. It is a thin wrapper encapsulating other MAC protocols implemented in different stacks. We further propose three abstractions, i.e., `property`, `services`, and `events` to describe each MAC's native functionalities. Property means public data or parameters which can be set or get by other modules. Services mean functions that can be invoked by other modules while events mean callback functions so that other modules can be notified of the occurrence or completion of the corresponding events. Figure 4 (a)–(c) show example data structures for 15.4 LPL MAC, BLE MAC and LoRaWAN. There are common fields such as kind, name, and MTU size. There are also specific fields for each different MACs. For example, for 15.4 LPL MAC, `localWakeupInterval` is used to control the duty cycle of the current node while the `remoteWakeupInterval` is used to set the preamble length in order to wakeup the receiver. For BLE MAC, parameters such as `scanInterval`, `advertiseInterval` are used to control the BLE MAC specific behaviors.

Second, TinyNet's MAC unification layer intends to present MAC independent interfaces to upper layers. TinyNet has three MAC adaptation modules: BLE adaptation, LoRaWAN adaptation, and LPL adaptation. These modules encapsulate radio specific functionalities and provide unified interfaces, i.e., uniMAC, to upper layers. Figure 4(d) shows example data structure for uniMAC which implements essential functions at layer 2.5. It is worth noting that as long as we implement the uniMAC interface for other radio technologies, we can incorporate those radios into our TinyNet architecture. On the other hand, the uniMAC interface does not prohibit the MAC-specific optimization. The upper layer protocols can still invoke the MAC-specific interfaces for this optimization.

4.2 Packet queue

The packet queue module is responsible for buffer management and packet scheduling. TinyNet provides FIFO packet scheduling and

```
struct lpl802154mac{

  kind_t kind;//lora, BLE or 15.4
  uint8_t name, channel, mtu;
  uint8_t localWakeupInterval;
  uint8_t remoteWakeupInterval;
  ...
  msg_t  *msg;
/***********service**********/
  uint8_t (*init)(void);
  uint8_t (*send)(msg_t *msg);
  uint8_t (*getLqi) (msg_t *msg);
  uint8_t (*getRssi) (msg_t *msg);
  ...
/***********event***********/
  void (*sendDone)(uint8_t
    evtType, msg_t *msg);
  ...
};
```

(a) 15.4 LPL MAC

```
struct blemac{
/***********property*********/
  kind_t kind;//lora, BLE or 15.4
  uint8_t name, channel, mtu;
  uint16_t scanInterval;
  uint16_t advertiseInterval;
  ...
  blem_t  *msg;
/***********service**********/
  uint8_t (*init)(void);
  uint8_t (*send)(blem_t *sdu);
  uint8_t (*adv)(addr_t *addr);
  uint8_t (*getRssi)(blem_t *msg);
  ...
/***********event***********/
  void (*sendDone)(uint8_t
    evtType, blem_t *msg);
  ...
};
```

(b) BLE MAC

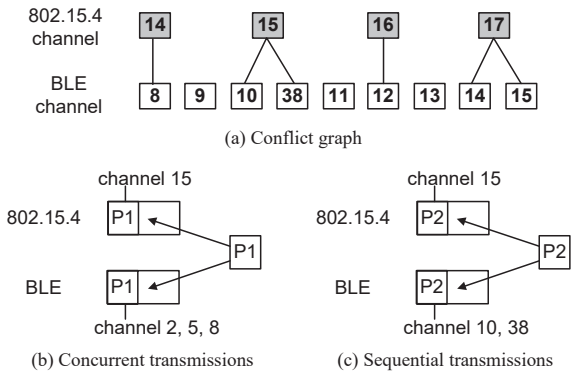
```
struct lorawanmac{
/***********property*********/
  kind_t kind;//lora, BLE or 15.4
  uint8_t name, channel, mtu;
  class_t class;
  uint8_t joinType;//ABP or OTAA
  ...
  lmsg_t *msg;
/***********service**********/
  uint8_t (*init)(void);
  uint8_t (*send)(lmsg_t *msg);
  uint8_t (*join)(uint8_t
    joinType);
  ...
/***********event***********/
  void (*sendDone)(uint8_t
    evtType, lmsg_t *msg);
  ...
};
```

(c) LoRaWAN MAC

```
struct unimac{
/***********property*********/
  kind_t kind;//lora, BLE or 15.4
  uint8_t name, channel;
  uint16_t mtu;
  ...
  unimsg_t *msg;
/***********service**********/
  uint8_t (*init)(void);
  uint8_t (*send)(unimsg_t *msg);
  uint8_t (*getLinkQuality)(
    unimsg_t *msg);
  ...
/***********event***********/
  void (*sendDone)(uint8_t
    evtType, unimsg_t *msg);
  ...
};
```

(d) uniMAC

Figure 4: Example data structures for 15.4 LPL MAC, BLE MAC, LoRaWAN and uniMAC.



(a) Conflict graph

(b) Concurrent transmissions

(c) Sequential transmissions

Figure 5: Multi-radio packet scheduling in TinyNet. (a) Conflict graph. (b) an example for concurrent transmissions. (c) an example for sequential transmissions.

priority-based packet scheduling for the single radio stack. TinyNet also provides a unique conflict graph-based packet scheduling approach for multi-radio platforms to avoid transmitting on the same ISM band.

Conflict graph-based packet scheduling: For an IoT node with n radios, n packet queues are required, one for each radio. When a packet is to be transmitted, TinyNet first determines which radios it should be transmitted on. It is possible that a packet will be transmitted on all radios, e.g., when a gateway with multiple radios broadcasts packets to all IoT nodes with heterogeneous radios. In an ideal case, TinyNet can transmit packets in different packet queues *concurrently* since TinyNet is currently based on multi-radio platforms. However, it is not always appropriate since there could be interference if the radios operate in the same frequency band and the selected channels overlap with each other. For example, 802.15.4 and BLE both operate on the 2.4GHz ISM band. 802.15.4 channel 15 overlaps with BLE channel 10 and 38. Therefore, the above 802.15.4 channel and BLE channels cannot be used concurrently.

TinyNet adopts the following approach (see Figure 5) to maximize concurrency while avoiding conflicts.

- There is a conflict graph where nodes represent different channels of each radio and an edge between a pair of nodes represents a conflict in between.
- If a packet is to be transmitted, it is pushed to appropriate queues. For the example shown in Figure 5, packet P1 is pushed to both the 802.15.4 queue and BLE queue since it should be transmitted on both radios.
- The above push operation triggers the packet transmission events. If the channels of two packets overlap, TinyNet avoids concurrent transmission by setting the busy flag in the first transmission event and avoids entering the second transmission event by checking the busy flag.

There are two points worth emphasizing here. First, TinyNet tries to avoid external WiFi interference. It selects 802.15.4 channel 15 which does not overlap with any WiFi channels. In addition, it relies on BLE's built-in channel hopping mechanism to avoid interference with WiFi. Second, our multi-radio packet scheduling algorithm is scalable for incorporating more radios (e.g., WiFi at 2.4GHz, LoRa 2.4GHz, etc.) in the future. The basic scheduling algorithm remains the same as long as the conflict graph is revised to consider more radio channels.

4.3 Neighbor management module

TinyNet provides a *unified* neighbor table with the following format: neighbor address, radio type, radio-on time, radio-off time, link quality, and channels. The channel entry is used for channel hopping based protocols for recording the synchronized transmission channels.

A key function provided by neighbor management is neighbor discovery. For single channel based protocols such as 802.15.4, neighbor discovery is relatively simple and we can borrow existing approaches for Low-Power-Listening based protocols [41]. The design for channel hopping based protocol is challenging considering flexibility, code reuse, compatibility, and efficiency.

For example, the BLE built-in neighbor discovery mechanism has limitations that require a redesign. There are two roles, i.e., central and peripheral in BLE. A central can discover non-connected peripherals in its neighborhood. A peripheral broadcasts packets with a default advertising period of 20ms. In one advertisement duration, three packets are transmitted consecutively on channels 37, 38, and 39. A central periodically performs channel scans, i.e., listening on one advertising channel in turn, to receive advertising

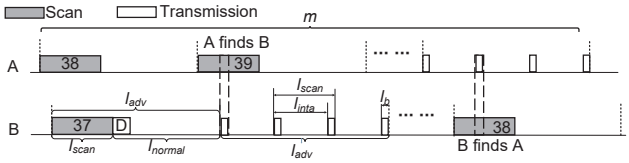


Figure 6: Channel hopping based neighbor discovery in TinyNet. Two nodes can discover each other when node A performs channel scan while node B is advertising.

packets from potential peripherals. However, when a peripheral connects to a central, it can no longer discover other central or peripheral devices. Similar to BLE, LoRaWAN also intends for single-hop communications and uses channel hopping mechanisms. Different from BLE, there is no neighbor discovery in LoRaWAN since the gateway is able to listen and receive packets at all channels.

To address the above limitation, we adopt a *fast bi-directional neighbor discovery* for channel hopping based protocols including BLE and LoRaWAN. An ordinary IoT node, say i (e.g., BLE peripheral or LoRaWAN client), has three operating modes including channel scan, normal operation, and advertisement.

- The channel scan mode is used for node i to discover other neighboring nodes and its duration is denoted as l_{scan}.
- The normal operation is used for normal data transmission (e.g., connection events in BLE) or sleep to save energy. The normal operation mode always starts after the channel scan mode. Its duration is denoted as l_{normal}. A normal time slot is used for channel scan followed by normal operation and its duration equals to l_{scan}+l_{normal}.
- The advertisement mode is used in order to let others to discover node i. In the advertisement mode, a node can periodically broadcast advertising packets with different channels. An advertising time slot is used for advertisement and its duration is denoted as l_{adv}. ($l_{\text{adv}} = l_{\text{scan}}$+$l_{\text{normal}}$).

For an ordinary node i, an advertisement time slot appears every m (e.g., $m = 10$) time slots as shown in Figure 6. If the advertisement time slots of two nodes appear at the same time, the two nodes cannot discover each other. The neighbor discovery module in TinyNet tries to avoid this situation by randomly allocating the positions of advertisement time slots.

The above approach has the following features. First, it inherits the design for BLE and thus can reuse a portion of code in its neighbor discovery module. Second, it enhances the original BLE neighbor discovery in the sense that an ordinary node can discover neighboring nodes and it can also be discovered by other nodes. At the same time, it is *compatible* with the original BLE neighbor discovery mechanism, i.e., a legacy non-connected BLE peripheral can be discovered and connected to TinyNet-enabled BLE nodes. Finally, it works for both BLE and LoRaWAN with different configurations.

4.4 Packet relay module

The packet relay module is important to enable multi-hop wireless routing. For 802.15.4 radio, packet relay is relatively simple since it uses a single channel. For BLE and LoRaWAN, however, there is no direct support for packet relaying since both radios are originally targeted for single-hop communication scenarios.

TinyNet employs an *automatic synchronization approach* for channel hopping based protocols in order to let the relay node receive packets from downstream nodes. Since both BLE and LoRaWAN use channel hopping, it is necessary to allocate reception slots and synchronize transmission/reception behaviors between the sender and the receiver. TinyNet automatically allocates the reception slots during rendezvous points where the downstream nodes perform channel scan and the relay node performs advertising. The downstream nodes can request one or more transmission slots (or a connection event with variable duration in the BLE term), and the relay node coordinates the allocation and confirms the final decision to the downstream nodes.

In order to optimize the performance for multiple continuous packets, which is common for the case when a large packet is fragmented into many smaller ones, there is a default *delay-after-receive* mechanism for all three radio technologies. A relay node will stay in the receiving state for a while after receiving a packet, attempting to receive more subsequent packets. However, there is a limit, i.e., it cannot exceed the allocated time slot or the length of a connection event.

5 Protocol Composition

In this section, we show that various existing protocols can be composed from the modules provided by TinyNet. In addition, we can also compose new protocols for better performance. Table 2 provides a summary of two existing protocols and two new protocols as well as their corresponding modules.

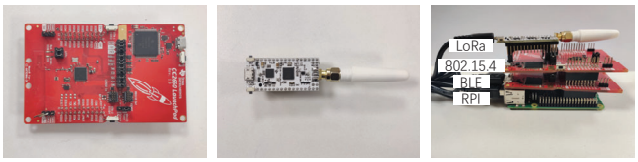
RPL over BLE. ALBER [31] is an adaptation layer for enabling RPL over BLE. RPL is a multi-hop wireless routing protocol based on 6LoWPAN which compresses IPv6 headers to meet the resource constraints of IoT nodes. For header compression, the packet fragmentation module and the header compression module are necessary. The header compression module also provides functionality for packet decompression. To provide multi-hop routing in RPL, neighbor discovery is required. Channel hopping based neighbor discovery is selected since ALBER is designed for BLE which is channel hopping based. We use RTT-based link estimation for ALBER. The packet relay module is necessary for forwarding packets. The packets are scheduled in FIFO. ALBER reuses BLE MAC layer, and therefore the basic unicast and broadcast services are used.

SP. SP [43] is an L2.5 layer protocol designed for 802.15.4 based sensor networks. SP allows link layers and network protocols to cooperate by maintaining and exposing a shared neighbor table and message pool. Priority-based packet scheduling is utilized so that urgent packets, e.g., control packets, can get higher priority in SP. SP uses the basic unicast and broadcast services.

Rateless BLE. TinyNet allows us to create protocols with coding support which is especially important in noisy environments. Rateless BLE provides rateless coding support in BLE communications so that BLE communication can be more robust with external interference. We use the LT codes implementation in the link layer coding module. Reliable transmission services for unicast, multicast and broadcast are used for packet retransmissions. The multicast group addressing module is required to support multicast transmission and reception.

Table 2: Decomposition of existing and composition of new network protocols.

Protocol	Fragment/ Hdr Compress		Packet Transmission	Neighbor Management	Addressing Module	Link Layer Coding	Network Layer
Existing Protocols							
RPL over BLE [31]	Fragment + Hdr Compress	Packet Relay	FIFO	Link Est. + Neighbor Dis.	Unicast Addressing	–	RPL+IPv6
SP [43]	–	–	Packet Priority	Link Est. + Neighbor Table	Unicast Addressing	–	–
New Protocols							
Rateless BLE	–	–	Reliable Transmission	FIFO	–	Multi. Group Addr. + Uni. Addr.	Rateless Code	–
RPL over LoRaWAN	–	Packet Relay	Uni./Broad.	FIFO	Link Est. + Neighbor Dis.	Unicast Addressing	–	RPL+IPv6



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

Figure 7: IoT nodes for the implementation.

RPL over LoRaWAN. With TinyNet, we can not only compose ALBER which enables multi-hop routing based on BLE, but also compose new protocols enabling multi-hop wireless routing based on LoRaWAN. RPL-over-LoRaWAN uses RPL for multi-hop routing. Therefore, the IPv6 and RPL modules are selected. We can use PRR based link estimation since the underlying MAC protocols in LoRaWAN provides packet reception status. For neighbor discovery, we use channel hopping based neighbor discovery (Section 4.3) since LoRaWAN is also channel hopping based. The packet relay module is required in order to allocate the time slots for packet transmission and reception.

6 Implementation

Hardware and OSes. We implement TinyNet on CC2650 (ROM 128kB, RAM 20kB, multi-standard radio chip supporting 802.15.4/ZigBee and BLE) and Heltec IoT LoRa node 151 (ROM 256kB, RAM 32kB). We have also built a gateway comprising of a Raspberry Pi 3 (RPI) [44] connected with two CC2650 chips and one Heltec IoT LoRa chip. Figure 7 shows these three types of nodes.

We have originally implemented TinyNet on Contiki OS 3.0 since Contiki OS has a lightweight implementation of protocols TinyNet relies on, e.g., 6LowPAN, uIP. To see the portability of TinyNet, we have also ported our implementation to RIOT [15] which also supports a rich set of protocols.

Implementation on Contiki OS. We reuse some existing works in TinyNet, e.g., BLEach [47] for BLE MAC, uIP [10] stack for the network layer and transport layer (see Table 3). New modules are implemented in TinyNet to support a wide range of radios. For example, the packet relay module schedules the transmission/reception slot for downstream/upstream nodes to forward packets. To this end, TinyNet extracts the timer variable in BLEach and provides a unified interface to L2.5. To let BLE nodes can discover each other even when they are connected, we also need to carefully deal with the parameter confliction (e.g., hopped channel), and reschedule states of BLE, e.g., connecting the scan state to the advertisement state. About 2,600 lines of C codes are added to implement the new modules at the L2.5.

Table 3: Module implementation in Contiki and RIOT

	Contiki OS	RIOT
Reused modules	uIP [10], RPL [12], Sicslowpan [12], RDC [12], BLE MAC [47]	GNRC [18], GNRC LoRaWAN [18], BLE MAC(Nimble [3])
Modified modules	BLE adaptation, Addressing module, LPL MAC, LoRaWAN MAC	Addressing module, LPL MAC
Newly added modules	Link layer coding, Packet Relay, Neighbor management, Packet queue, LoRaWAN adaptation, LPL adaptation	Packet relay, Packet queue, Neighbor management, RDC, Link layer coding, LPL adaptation, BLE adaptation, LoRaWAN adaptation

Table 4: Code size and memory footprint (B) of TinyNet on CC2650 with support for BLE and 802.15.4

	Contiki OS	RIOT
Total code size	60,434	95,445
Total memory (static)	8,174	10,632
Total memory (dynamic)	0~6,162	0~5,372

Implementation on RIOT. For the implementation on RIOT, we reuse its GNRC protocol stack and NimBLE stack (see Table 3). We need to modify the addressing module and LPL module (which was originally designed for TinyOS [41]) to make them work. The newly added module, e.g., packet relay, packet queue and neighbor discovery, can mostly be adapted from the implementation on Contiki OS. About 1,500 lines of codes are required to make TinyNet work on RIOT. Table 4 shows the code size and memory footprint for the implementation of TinyNet on CC2650 with support to two radio technologies—802.15.4 and BLE.

Optimization techniques on IoT devices. We choose existing lightweight implementations for TinyNet, e.g., uIP in Contiki OS, BLE MAC in BLEach [47]. We have employed zero copy technique [58] to save memory consumption: A buffer named *packetbuf* is used to pass through data packets from the network layer to L2.5. A circular buffer is used for the packet receive queue to re-assemble packets that arrived out of order.

Special considerations for the gateway. We have built a gateway node comprising RPI and three radio chips. The hardware performance of RPI (1.2 GHz CPU, 1 GB RAM [44]) is generally more powerful than the radio devices (48 MHz CPU, 28 kB RAM [6]). TinyNet is currently designed to let L2.5 and the above layers run on the RPI, while the link layer and the lower layers run on the radio devices. RPI and the radio devices are communicated with

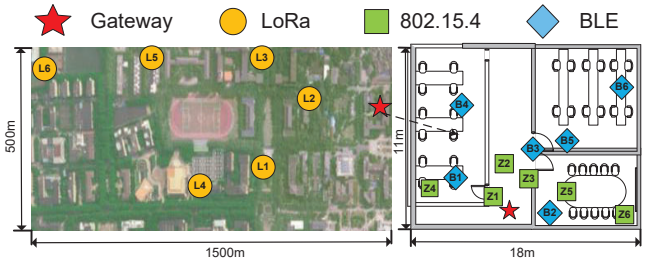


Figure 8: A mixed indoor and outdoor testbed consisting of heterogeneous IoT nodes.

Table 5: Notations for different network topologies

Notation	Network topology	Notation	Network topology
ZGB	↔ ↔	ZGBB	↔ ↔ ↔
ZGL	↔ ↔ LoRa	ZGLL	↔ LoRa ↔ LoRa
BGL	↔ ↔ LoRa	—	—

a standard protocol, i.e., Serial Line Internet Protocol (SLIP) [1]. Multiple radio devices are connected through the serial ports and maintained by the co-thread mechanism of Contiki OS. It is worth noting that our implementation can also be ported to COTS multi-radio gateways which are already on the market, e.g., Samsung Connect Home [45], Google WiFi AP [19], and Cisco Catalyst Series [7].

7 Evaluation

In this section, we perform a systematic evaluation of TinyNet. We compare TinyNet with two existing IoT network stacks, i.e., Tencent stack implemented in TencentOS Tiny [49] and GNRC implemented in RIOT [15]. We have mainly used a mixed indoor and outdoor testbed for the evaluation (See Figure 8). We deploy the gateway node in our laboratory. Six LoRa nodes are deployed outdoor in a 500m by 1500m area on the roof of buildings. Six BLE and six 802.15.4 nodes are deployed indoors. The transmission power of IoT nodes can be configured to form different topologies. Table 5 shows the notation of each scenario and the corresponding network topology. Different IoT nodes communicate with each other using different protocols, e.g., UDP and TCP. When not specified, the data transmission is one packet every five seconds with a user payload length of 50 bytes. We record packets transmission and reception timestamps for the later analysis.

7.1 Interoperability

TinyNet can enable new interoperatibility scenarios that previous works cannot easily provide. In this subsection, we provide representative use cases of TinyNet in different scenarios.

ZigBee-BLE communication using TCP. Figure 9 shows the use cases of TCP among two CC2650 (configured to use ZigBee and BLE, respectively) across a gateway. These two nodes, although use different radio technologies, can use the standard socket interface, e.g., connect(), accept(), send(), recv(). The round-trip time is roughly 275.2ms.

WiFi-BLE using CoAP and RESTful API. Figure 10 shows the use case of RESTful API over CoAP. In this case, a CC2650

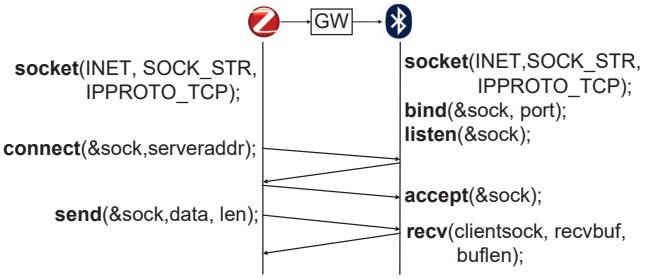


Figure 9: ZigBee-BLE communication using TCP.

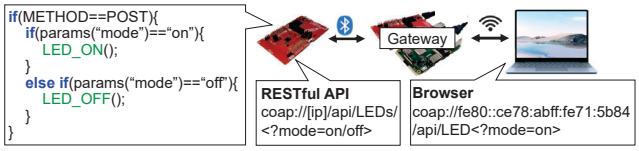


Figure 10: WiFi-BLE using CoAP and RESTful API.

node uses BLE to connect to a gateway (RPI) which turns on the WiFi hotspot and thus can transmit or receive WiFi packets with a laptop. In this case, a laptop or smartphone can turn on/off an LED on CC2650 across a gateway as long as the IoT node implements and exposes the RESTful API. In the future, when IPv6 is supported by the global Internet, a laptop/smartphone, no matter where it is, can remotely control IoT devices supporting TinyNet. The round-trip time is roughly 242.1ms which is slightly lower than TCP. This is because CoAP, although inspired by HTTP, was designed to use UDP instead of TCP.

ZigBee-BLE, LoRa-ZigBee, LoRa-BLE using TCP, UDP, and MQTT. We evaluate the delay of TinyNet in the above different scenarios. In all scenarios, the sender sends a packet of 50 bytes payload to the receiver per second. We turn the radio always on for the 802.15.4 link and the connection interval of BLE is set to 50 ms. We have also compared with the *vertically-integrated approach* in which nodes transmit/receive link-layer packets to/from the gateway and the gateway performs protocol conversion.

From Figure 11(left), we can see that: (1) TinyNet allows interoperability among heterogeneous nodes at the network layer and the above layers. (2) The delays of TinyNet-TCP are only slightly higher than those of TinyNet-UDP. This is because that the TCP connection is maintained during the entire experiment and the connection establishment overhead is amortized to multiple packets. (3) Compared with the MAC+protocol conversion approach, all approaches of TinyNet introduce additional delays due to the overhead of processing upper-layer protocols. Specifically, the delay increase of TinyNet-TCP is 13.1% which is comparable with 15.9% delay increase of prior work—TCPlp [30] over 802.15.4 links. For traffic from/to LoRa, the delay overhead is larger since LoRa has a much lower data rate than ZigBee and BLE.

Multihop wireless communication using MQTT. Figure 11 shows the delay performance from an 802.15.4 node to other nodes several hops away through a gateway. We see that: (1) TinyNet enables communication via MQTT among heterogeneous nodes, even if they are several hops away. (2) The communication delay increases with the increasing number of hops.

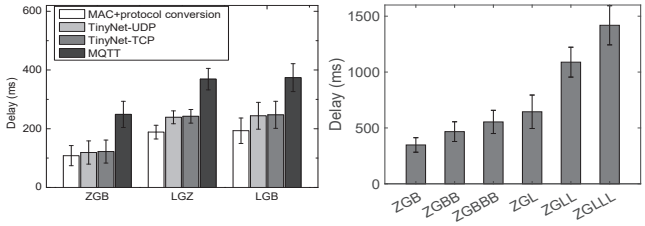


Figure 11: Left: Delay between heterogeneous IoT nodes through a gateway. Right: MQTT delay from a 15.4 node to BLE/LoRa nodes multiple hops away. See Table 5 for notations in the x-axis.

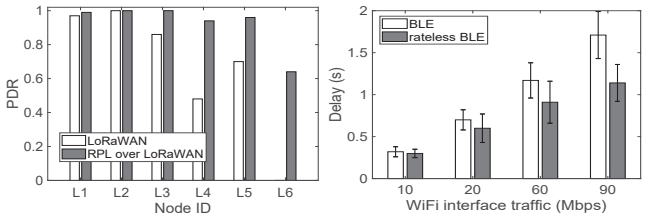


Figure 12: Left: Packet delivery ratio (PDR) for different nodes. Right: Delay for BLE and rateless BLE.

Summary. In IoT, heterogeneous radio protocols co-exist today, making the integration of data and services from various devices extremely complex and costly [21]. Bringing IP to each device makes the integration across systems and applications much simpler, regardless of the underlying radio technologies and network topologies. TinyNet enables universal connectivity between heterogeneous devices using standard socket interfaces as well as the upper-layer web technologies.

7.2 Benefits of modularity and new protocol composition

We evaluate the performance of two new protocols which can be easily created by composing different modules in TinyNet.

RPL over LoRa. Recent studies [34] have shown that the LoRa's typical communication range is still limited to 0.1-2 km, due to many practical factors such as obstacles, multipath fading, etc. Multihop communication is very useful to further extend LoRa's communication range.

We conduct an experiment using the testbed shown in Figure 8. Each LoRa node sends data packets to the gateway every two minutes for 15 days. In this process, we adopt the default parameter settings for all LoRa nodes, e.g., the transmission power is 13 dBm. Figure 12(left) depicts the end-to-end packet delivery ratio (PDR) for two protocols, i.e., the original LoRaWAN without multihop support and the newly created "RPL over LoRa" protocol by composing different TinyNet modules. We can see that: (1) With LoRaWAN, the PDRs of individual nodes are relatively low: The PDR of node 6 even drops to 0%, indicating that node 6 cannot communicate with the gateway directly. (2) With RPL over LoRa, the PDRs can be significantly increased. For example, the PDR of node 6 increases to 64%. It is very useful that TinyNet's modular design allows reusing RPL for LoRa. The provided multi-hop communication capability can significantly increase PDR without deploying additional gateways.

Rateless BLE. With the increase of radio technologies, cross-technology interference becomes a critical issue [22]. Rateless coding is a common approach to achieving robust communication in such situations. We conduct experiments based on the same testbed. The gateway transmits one packet to the BLE nodes every 10 seconds in a multicast manner. Note that the multicast capability of BLE is accomplished by using the advertising channel and the addressing engine. We use a laptop placed close to the gateway to generate WiFi traffic to interface with the BLE transmissions. On the laptop, we use iperf3 to generate data traffic with varying bit rates.

Figure 12(right) shows the average transmission delays under varying WiFi bit rates. The transmission delay is measured as the time duration between the sending time of a packet at the gateway and its reception time at all BLE devices. We can see that: (1) The transmission delay increases with the interference increases due to the more packet retransmissions. (2) Rateless BLE reduces transmission delays, especially under heavy interference. For example, when the WiFi bit rate is set at 90 Mbps. The average transmission delay of rateless BLE is 1.1s, compared with 1.7s for the original BLE, resulting in a 35% reduction.

7.3 Code size and memory footprint

We quantify TinyNet's code size and memory footprint on CC2650 and Heltec LoRa node 151 with its default implementation on Contiki OS. Code size refers to the amount of program memory (ROM) occupied by the networking code, while memory footprint refers to the amount of RAM allocated. Both metrics are important for IoT nodes. Several parameters may influence the memory usage, e.g., the number of connected radio devices at the gateway, the capacity of packet queue, etc. In the current evaluation, we set the common settings according to existing works [43, 47], e.g., the capacity of the packet queue is set to 36.

Table 6 shows each module's code size and static memory footprint for the full implementation of TinyNet on the gateway node. We can see that: (1) For the implementations for a single radio, TinyNet consumes 6.146–6.588 KB in RAM and 48.013–51.630 KB in ROM. (2) For the implementation of three radios, it consumes up to 10.367 KB in RAM and 77.411 KB in ROM.

Comparative study. Table 7 shows the code size and memory footprint for several existing protocols and new protocols, using the monolithic approaches (including Contiki OS uIP, Tencent stack, and RIOT GNRC) and TinyNet's modular approach. These approaches implement the protocols on the same node (i.e., Raspberry Pi 3) for a fair comparison. For the existing protocols, "802.15.4 6LP + BLE 6LP" refers to the implementation of 6LoWPAN over 802.15.4 and BLE, which uses a separate implementation for the two radio technologies.

Three observations can be made from Table 7: (1) For protocols for a single radio (i.e., Contiki uIP and RPL-over-BLE), monolithic approaches and TinyNet's modular approach result in *comparable* code size and memory footprint. Compared with the monolithic approaches, TinyNet has an extra overhead to allow demultiplexing and configuration among different modules. (2) For the protocol for multiple radios (i.e., 802.15.4 6LoWPAN and BLE 6LoWPAN),

Table 6: Code size and memory footprint of TinyNet.

Module	Code size (kB)	Memory (kB)
Transport layer		
TCP	4.084	0.221
UDP	0.962	1.297
Network layer		
IPv6	11.696	1.744
RPL	11.39	0.32
2.5 layer		
Packet relay	0.486	0.085
Fragmentation & Compression	3.231	1.051
Addressing module	0.134	0.027
Link layer coding	1.523	0.742
Data Tx & Rx	0.370	0.064
Packet queue	0.312	0.256
Neighbor management	0.543	0.172
Adaptation	1.266	0.489
MAC layer		
BLE MAC	6.750	1.089
LPL MAC & RDC	8.503	0.955
LoRaWAN MAC	11.270	0.903
PHY layer		
BLE PHY	6.527	0.873
802.15.4 PHY	6.052	0.694
LoRa PHY	5.734	0.682
Total		
TinyNet for BLE	48.013	6.588
TinyNet for 802.15.4	48.924	6.146
TinyNet for LoRa	51.630	6.222
TinyNet for multi-radios	77.411	10.367

Table 7: Code and memory size comparison of the monolithic approach and TinyNet's modular approach.

Protocol		Code Size (kB)		Mem. Footprint (kB)	
		Mono	TinyNet	Mono	TinyNet
Existing Protocol	Contiki uIP	11.76 (uIP)	12.27	1.74 (uIP)	1.78
	RPL over BLE	10.99 (uIP) 11.01 (GNRC)	11.39	0.30 (uIP) 1.62 (GNRC)	0.32
Co-exist. Protocol	802.15.4 6LP+ BLE 6LP	8.40 (uIP) 8.46 (Tenc.) 9.32 (GNRC)	5.08	3.52 (uIP) 1.43 (Tenc.) 1.41 (GNRC)	1.27
New Protocol	RPL over LoRaWAN	-	43.14	-	5.97
	Rateless BLE	-	15.98	-	3.22

TinyNet achieves *significant* reduction. Specifically, TinyNet occupies 39.5%–45.5% less code size, and 9.9%–63.9% less memory footprint compared with the three monolithic approaches. (3) For new protocols (e.g., RPL over LoRaWAN, and rateless BLE), it is expected that the code and memory sizes are larger than those of the original protocols without the new functionalities.

7.4 Communication delay

Comparative study. We perform a comparative study on the delay performance. We use BGL (see Table 5) for the network topology. One BLE node sends a 20-byte packet to the LoRa node every second. Tencent stack and GNRC have not implemented the TCP/IP protocols upon LoRa. Therefore, we perform necessary protocol conversions. Figure 13 shows the delays of TinyNet, Tencent stack and GNRC between the BLE node and the LoRa node. Results show that TinyNet achieves the best performance. This is because the gateway does not have to wait for the uplink packets to perform the downlink transmissions over the LoRa link thanks to TinyNet's automatic synchronization approach for allocating transmission/reception slots.

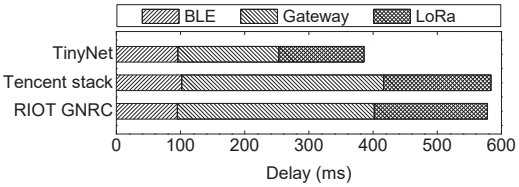


Figure 13: Delay comparison with the existing work.

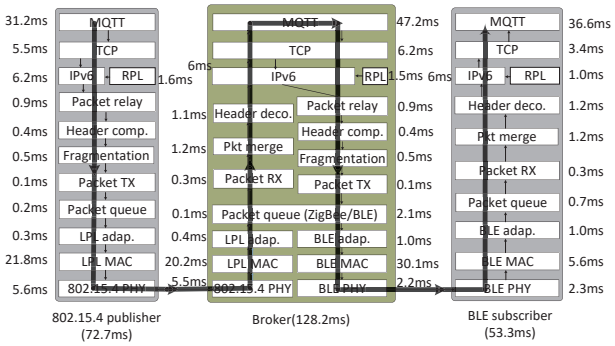


Figure 14: Delay breakdown. An example of the 802.15.4 node publishes MQTT messages to the BLE node through the multiradio gateway.

Breakdown. We present the delay breakdown in the ZGB topology using the MQTT protocol. In this case, An MQTT publisher transmits a 50-byte message to the gateway that is running the MQTT broker. The published message is distributed by the gateway to the subscribers that register the corresponding topic. We turn the radio always on for the 802.15.4 link and the connection interval of BLE is set to 50 ms. Although we use a specific topology for connecting BLE and 802.15.4, note that TinyNet's design allows that the MQTT publisher/subscriber can be any of the three radio technologies, i.e., BLE, LoRa and 802.15.4.

Figure 14 presents an example breakdown for the publisher (802.15.4), the MQTT broker (gateway with 802.15.4 and BLE) and the subscriber (BLE). The results are averaged from 300 MQTT packets containing the same payload. We can see that the processing delay on the gateway contributes the most to the end-to-end delay, i.e., almost 51.5% of the overall delay.

7.5 Energy efficiency

We evaluate TinyNet's energy efficiency in terms of radio duty cycle. This is reasonable since the radio-on time largely determines the lifetime of most IoT nodes [16].

We perform an evaluation in the BGB and ZGZ topology. The packet payload size is 100 bytes and the sending interval is one second. In the BGB scenario, a BLE node sends TCP packets to another BLE node with different connection intervals. In the ZGZ scenario, two 802.15.4 nodes use LPL MAC with different sleep intervals.

We compare the radio duty cycle of TinyNet with Tencent stack and GNRC. As a baseline, we also compare with the MAC+protocol conversion approach, i.e., the devices only use the original BLE and 802.15.4 LPL MAC over the links and rely on the gateway for link layer protocol conversion. Figure 15 and Figure 16 show the results.

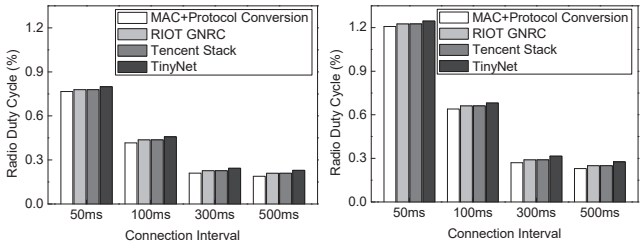


Figure 15: Energy consumption comparison in the BGB scenario. Left: sender. Right: receiver.

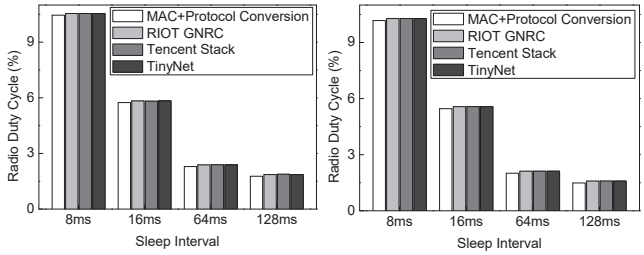


Figure 16: Energy consumption comparison in the ZGZ scenario. Left: sender. Right: receiver.

We can see that (1) TinyNet, Tencent stack and RIOT GNRC result in the similar radio duty cycles, which are slightly larger than the baseline. (2) Compared with Tencent stack and RIOT GNRC, TinyNet has slightly larger duty cycles due to additional functionalities like bidirectional neighbor discovery. (3) In the BGB scenario, the radio duty cycle decreases when the connection interval increases. It is expected because BLE requires control packet exchanges every connection interval and the overhead of control packets decreases with increasing connection interval. (4) In the ZGZ scenario, the radio duty cycle decreases when the sleep interval increases. This is because the frequency of CCA (Clear Channel Assessment) decreases with the increasing sleep interval.

8 Discussion

We focus now on what can be distilled from the design and implementation of TinyNet. We first revisit the principles TinyNet builds upon. We then emphasize that there are important design techniques employed in TinyNet, but which are different from the earlier sensor networking systems or other more general-purpose systems. We finally contrast the important aspects of TinyNet and sensor network systems.

8.1 Principles TinyNet builds upon

L2.5 designs. We leverage the layering concept from networking protocols, enabling scalability to multiple radio technologies and thereof unification. Similar to SP [43], we also introduce an abstraction layer L2.5 in TinyNet to provide a *unified* interface on top of a wide range of data link and physical layer technologies that allow the network layer and the upper-layer protocols to operate efficiently through link independent optimizations. Although the principles are similar, the detailed design considerations are quite different. For example, TinyNet has more modules at L2.5. Some modules, e.g., fragmentation and header compression, are necessary for supporting 6LoWPAN and IPv6. Some other modules, e.g., BLE adaptation and LoRaWAN adaptation, are necessary for unifying different radio technologies. Besides, some key modules in TinyNet have drastically different designs as described in Section 3.

Modular approach. Modularity is a well-known approach that eases the implementation of new protocols by increasing code reuse, and enables co-existing protocols to share and reduce code and resources consumed at run-time, fostering greater intellectual synergy. TinyNet's modular designs differ from previous works [13, 29] in two ways. First, TinyNet provides a different set of modules supporting key services including (1) unification of different radios (2) interoperability (3) reliability consideration for low-power links. Second, unlike TinyOS's static approach and compile-time wiring, TinyNet makes use of pointers and dynamic memory allocations, allowing adding more advanced features (e.g. dynamic loading of the modules) in the future.

8.2 Techniques TinyNet employs

uniMAC abstractions. In TinyNet, uniMAC provides a general interface that encapsulates many low-level details of different radio technologies. While some functions are direct encapsulation of the underlying layer, the design of other functions requires special consideration for different radio technologies. For example, layer 3 protocols, such as RPL, usually require an API to get the link quality (e.g., `getLinkQuality()`) for routing selection. For low power and lossy networks, link quality metrics include RSSI, LQI, ETX, PRR, etc [2]. Link quality estimation may not be consistent among different MAC protocols, e.g., for LPL MAC, link quality may be overestimated due to the inaccurate counting problem of the wake-up packets, especially when bursty channel contention and coexistent interference appear [9]. As an adaptation layer, uniMAC translates these metrics to proper link quality values for the upper layer through calibration, normalization, combination, etc.

Unified neighbor management. Neighbor management is an essential function that all previous works provide. In TinyNet, we provide two major functionalities. First, we provide a unified neighbor table which provides services to many other modules. The neighbor management module differs from the existing ones [13, 32, 43]. It is designed for multiple underlying radio technologies, instead of a single one. Moreover, unlike the TinyOS network stacks where there are separate modules for link estimation, routing engine, and possibly topology management [51], TinyNet's neighbor management module integrates all the above functionalities. Second, we provide *bidirectional* and *continuous* neighbor discovery services. Bidirectional discovery means that one node can find other nodes and vice versa, which is important for integrating with the packet relay module in order to provide multi-hop communication capabilities. Continuous discovery is also important so that neighborhood information (e.g., neighbor identity, the corresponding link quality) can be regularly updated for dynamic routing selection.

Communication scheduling. While communication scheduling is a common and widely technique to improve transmission efficiency, TinyNet employs this technique in two different ways. First, we employ conflict graph-based scheduling for multi-radio platforms. Second, we use it for automatic synchronization of transmission/reception slots in the relay module. Synchronization

means alignment of time and frequency so that transmission can successfully arrive at the next hop. Unlike many TinyOS modules, e.g. FTSP [37], which perform *global* time synchronization, TinyNet exploits rendezvous points during data communication for *neighborhood* time synchronization. The neighborhood time information is then recorded and timely updated in the neighbor table. We find that neighborhood time information is sufficient for communication over a one-hop link, as well as a multihop path as communication over a path takes place hop by hop.

8.3 TinyNet vs. sensornet

Sensor networks are homogeneous systems deployed for an application-specific and collaborative purpose [32]. Due to limited communication range, sensornets typically employ multihop communications. The early wisdom is that end-to-end connectivity may not be the primary goal of sensornets. However, the landscape has shifted drastically in recent years with technological advances. More and more heterogeneous IoT devices are connected to the Internet. 802.15.4, BLE, and LoRa are all important techniques for IoT. It is beneficial to put TCP/IP over these low-power links so that IoT could be interoperable with the traditional TCP/IP networks to support popular IoT application protocols [30], like MQTT [40] and ZeroMQ and to simply IoT gateway design and to provide better services to the applications with high reliability and high link utilization requirements. It is also worth noting that we build TCP on top of BLE connected mode to provide end-to-end reliability. TCP's end-to-end reliability differs from the built-in BLE reliability mechanism which only protects a frame over a single link.

TinyNet intends to incorporate many standardized protocols into one architecture, providing end-to-end connectivity to the rest of the Internet. TinyNet focuses on the interoperability issue and tries to make IoT devices the first-class citizen of the Internet, while still encouraging innovations at the lower layers.

9 Related Work

TinyNet draws insights from many existing works. We divide them into three main categories as described as follows.

Network architectures for 802.15.4. SP [43] provides a unified interface to a wide range of link layer technologies that allows the network layer and the above protocols to operate efficiently through link-independent optimizations. Building on top of SP, NLA [13] (Network Layer Abstraction) proposes a modular network layer for sensor networks. This modularity eases the implementation of new protocols by increasing code reuse and enables co-existing protocols to share. The above existing architectures shed light on many important aspects in designing TinyNet. On the other hand, TinyNet makes several unique contributions. Specifically, TinyNet provides a unified L2.5 across different radio technologies. In contrast, all existing sensor network architectures focus on a single radio technology, i.e., 802.15.4.

Network architecture and protocols for BLE and LoRa. ALBER is [31] an adaptation layer between BLE and RPL, providing multi-hop support for BLE. It tightly couples RPL and BLE operations together. TinyNet can also adopt RPL at the network layer. However, TinyNet provides multi-hop support for a wide range of radio technologies. This consideration forces TinyNet to incorporate more generic modules. For example, to enable the multi-hop feature on LoRaWAN, TinyNet provides neighbor discovery which is absent in ALBER. BLEach [47] is a full-fledged IPv6-over-BLE stack. However, it is specifically designed for BLE while TinyNet is designed for many different radio technologies. LoRa is a relatively new radio technology which attracts significant research attention in recent years [5, 17, 27, 33, 52]. In [50], a LoRa-based networking stack is designed in order to enable standardized IPv6 LoRa communications. This stack only applies to the LoRa radio and does not consider code reuse in the implementation.

Interoperability for heterogeneous IoT networks. 6LoWPAN [53] was originally designed for sensor networks a decade ago. This standard specifies how to format IPv6 packets using a compact header over low-power wireless links. Some open-source IoT operating systems have options for supporting Internet protocols to some extent. For example, the default network stack in RIOT [15], i.e., GNRC, provides generic interfaces for supporting multiple heterogeneous interfaces and stacks that can concurrently operate. TencentOS Tiny [49] and Huawei LiteOS [23] provide the network stack support for different radios including 802.15.4, BLE, and LoRa. So far as we know, all the above stacks do not fully support interoperatability among common radio technologies. For example, RIOT's GNRC does not support IPv6 for LoRa. Besides research efforts, there is much industrial progress in recent years. Although most commercial-off-the-shelf (COTS) IoT devices are not interoperable with the Internet, this situation is rapidly changing. In 2019, Google, Apple, Amazon and the ZigBee Alliance have launched a project called CHIP [38] (i.e., Connected Home over IP, now changed to Matter) to simplify the connection between different IoT devices from different companies. At the core of this standard is the IP layer which provides interoperability between the IoT devices having different radio technologies including Bluetooth, Wi-Fi, Z-Wave, and ZigBee. Our work conforms to this trend and can provide better support for different radios, better modularity and better efficiency.

10 Concluding Remarks

In this paper, we design and implement TinyNet, a *lightweight*, *modular*, and *unified* network architecture for representative low-power radio technologies for the IoT. We implement TinyNet on two IoT OSes (Contiki OS and RIOT) and three types of IoT nodes. The work presented here is only a step towards an "complete IoT network architecture". Looking forward, there are multiple future research directions. First, more radio technologies and more network protocols should be incorporated. Second, some cross-layer issues should be carefully considered so that more tunable parameters will be exposed for cross-layer optimizations.

Acknowledgements

We sincerely thank our shepherd, Matt Welsh, and the anonymous reviewers for their valuable feedback. This work is supported by NSFC under grant no. 62072396, Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars under grant no. LR19F020001, the Fundamental Research Funds for the Central Universities (no. 226-2022-00087), and Alibaba-Zhejiang University Joint Institute of Frontier Technologies. Yi Gao is the corresponding author.

References

[1] A Nonstandard For Transmission Of IP Datagrams Over Serial Lines: Slip. 2020. https://tools.ietf.org/html/rfc1055. (2020).

[2] Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks. 2012. https://tools.ietf.org/html/rfc6551. (2012).

[3] Apache NimBLE. 2021. https://github.com/apache/mynewt-nimble. (2021).

[4] Bahl P, Adya A, Padhye J and Wolman A. 2014. Reconsidering wireless systems with multiple radios. In *ACM SIGCOMM Computer Communication Review*.

[5] Artur Balanuta, Nuno Pereira, Swarun Kumar, and Anthony Rowe. 2020. A cloud-optimized link layer for low-power wide-area networks. In *Proc. of ACM MobiSys*.

[6] CC2650 Launchpad Development Kit. 2020. http://www.ti.com/lit/ml/swru451/swru451.pdf. (2020).

[7] Cisco catalyst 9100 ap. 2021. https://www.cisco.com/c/en/us/products /wireless/catalyst-9100ax-access-points/. (2021).

[8] LoRa Alliance Technical Committee. 2017. LoRaWAN 1.1 Specification. *Standard V1* (2017).

[9] Mengshu Hou Daibo Liu, Zhichao Cao and Yi Zhang. 2016. Frame counter: Achieving accurate and real-time link estimation in low power wireless sensor networks. In *Proc. of IEEE IPSN*.

[10] Adam Dunkels. 2002. *uIP-A free small TCP/IP stack*. Technical Report.

[11] Adam Dunkels. 2011. The ContikiMAC Radio Duty Cycling Protocol. (2011).

[12] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. 2004. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Proc. of IEEE LCN*.

[13] Cheng Tien Ee, Rodrigo Fonseca, Sukun Kim, Daekyeong Moon, Arsalan Tavakoli, David Culler, Scott Shenker, and Ion Stoica. 2006. A modular network layer for sensorsets. In *Proc. of USENIX OSDI*.

[14] Rashad Eletreby, Diana Zhang, and et. al. 2017. Empowering Low-Power Wide Area Networks in Urban Settings. In *Proc. of ACM SIGCOMM*.

[15] Emmanuel Baccelli, Cenk Gundo gan, Oliver Hahm, Peter Kietzmann, Martine S. Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wahlisch. 2021. RIOT: The friendly Operating System for the Internet of Things. https://github.com/RIOT-OS/RIOT. (2021).

[16] Yasmin Fathy and Payam Barnaghi. 2019. Quality-Based and Energy-Efficient Data Communication for the Internet of Things Networks. *IEEE Internet of Things Journal* 6, 6 (2019), 10318–10331.

[17] Amalinda Gamage, Jansen Christian Liando, Chaojie Gu, Rui Tan, and Mo Li. 2020. LMAC: Efficient carrier-sense multiple access for lora. In *Proc. of ACM MobiCom*.

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

[19] Google WiFi AP. 2021. https://store.google.com/gb/product/ google_wifi_specs. (2021).

[20] Bluetooth Special Interest Group. 2014. *Bluetooth Specification Version 4.2*. Technical Report.

[21] Dominique D Guinard and Vlad M Trifa. 2016. *Building the web of things*. Vol. 3. Manning Publications Shelter Island.

[22] Anwar Hithnawi, Su Li, Hossein Shafagh, James Gross, and Simon Duquennoy. 2016. Crosszig: combating cross-technology interference in low-power wireless networks. In *Proc. of ACM IPSN*.

[23] Huawei LiteOS. 2020. https://www.huawei.com/minisite/liteos/en/. (2020).

[24] Jonathan W Hui and David E Culler. 2008. IP is Dead, Long Live IP for Wireless Sensor Networks. In *Proc. of ACM SenSys*.

[25] Internet Protocol, Version 6 (IPv6) Specification. 2020. https://tools.ietf.org/html/rfc2460. (2020).

[26] Hassan Iqbal, Muhammad Hamad Alizai, Ihsan Ayyub Qazi, Olaf Landsiedel, and Zartash Afzal Uzmi. 2018. Scylla: Interleaving Multiple IoT Stacks on a Single Radio. In *Proc. of ACM CoNEXT*.

[27] Kai-Hsiang Ke, Qi-Wen Liang, Guan-Jie Zeng, Jun-Han Lin, and Huang-Chen Lee. 2017. A LoRa Wireless Mesh Networking Module for Campus-Scale Monitoring. In *Proc. of ACM IPSN*.

[28] Kim, Hyung-Sin, Sam Kumar, and David E. Culler. 2019. Thread/OpenThread: A compromise in low-power wireless multihop network architecture for the Internet of Things. In *IEEE Communications Magazine*, Vol. 57. 55–61. Issue 7.

[29] Kevin Klues, Gregory Hackmann, Octav Chipara, and Chenyang Lu. 2007. A component-based architecture for power-efficient media access control in wireless sensor networks. In *Proc. of ACM SenSys*.

[30] Sam Kumar, Michael P Andersen, Hyung-Sin Kim, and David E. Culler. 2020. Performant TCP for Low-Power Wireless Networks. In *Proc. of USENIX NSDI*.

[31] Taeseop Lee, Myung-Sup Lee, Hyung-Sin Kim, and Saewoong Bahk. 2016. A synergistic architecture for RPL over BLE. In *Proc. of IEEE SECON*.

[32] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, and Eric Brewer†and David Culler. 2004. The Emergence of Networking Abstractions and Techniques in TinyOS. In *Proc. of USENIX NSDI*.

[33] Chenning Li, Hanqing Guo, Shuai Tong, Xiao Zeng, Zhichao Cao, Mi Zhang, Qiben Yan, Li Xiao, Jiliang Wang, and Yunhao Liu. 2021. NELoRa: Towards Ultra-low SNR LoRa Communication with Neural-enhanced Demodulation. In *Proc. of ACM SenSys*.

[34] Jansen C. Liando, Amalinda Gamage, Agustinus W. Tengourtius, and Mo Li. 2019. Known and Unknown Facts of LoRa: Experiences from a Large-scale Measurement Study. *ACM Transaction on Sensor Networks* 15, 2, Article 16 (Feb. 2019), 16:1–16:35 pages.

[35] Tyson Macaulay. 2016. *RIoT Control: Understanding and Managing Risks and the Internet of Things*. Morgan Kaufmann.

[36] Manyika J, Chui M, Bisson P, Woetzel J, Dobbs R, Bughin J, Aharon D. 2016. The Internet of Things: Mapping the Value Beyond the Hype. *arXiv e-prints* (2016).

[37] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. 2004. The Flooding Time Synchronization Protocol. In *Proc. of ACM SenSys*.

[38] Matter (formerly Project Connected Home over IP, or Project CHIP). 2022. https://github.com/project-chip/connectedhomeip. (2022).

[39] Mesh networking with the power of Bluetooth technology. 2020. https://www.bluetooth.com/specifications/mesh-specifications/. (2020).

[40] Message Queuing Telemetry Transport (MQTT). [n. d.]. http://mqtt.org/. ([n. d.]).

[41] Sunghyun Moon, Taekjoo Kim, and Hojung Cha. 2007. Enabling Low Power Listening on IEEE 802.15.4-Based Sensor Nodes. In *Proc. of IEEE WCNC*.

[42] Yao Peng, Longfei Shangguan, Yue Hu, Yujie Qian, Xianshang Lin, Xiaojiang Chen, Dingyi Fang, and Kyle Jamieson. 2018. PLoRa: A Passive Long-range Data Network from Ambient LoRa Transmissions. In *Proc. of ACM SIGCOMM*.

[43] Joseph Polastre, Jonathan Hui, Philip Levis, Jerry Zhao, David Culler, Scott Shenker, and Ion Stoica. 2005. A unifying link abstraction for wireless sensor networks. In *Proc. of ACM SenSys*.

[44] Raspberry Pi. 2020. https://www.raspberrypi.org/. (2020).

[45] Samsung Connect Home. 2021. https://www.samsung.com/sg/ smarthome/. (2021).

[46] Jianping Song, Song Han, Al Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt. 2008. WirelessHART: Applying wireless technology in real-time industrial process control. In *Proc. of IEEE RTAS*.

[47] Michael Spörk, Carlo Alberto Boano, Marco Zimmerling, and Kay Römer. 2017. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proc. of ACM SenSys*.

[48] Technical report of TinyNet. 2022. https://www.dropbox.com/s/ph5dqf59rncbfij/Technical%20report-TinyNet.pdf?dl=0. (2022).

[49] TencentOS Tiny: A real-time IoT system. 2020. https://github.com/Tencent/TencentOS-tiny. (2020).

[50] Steffen Thielemans, Maite Bezunartea, and Kris Steenhaut. 2017. Establishing transparent IPv6 communication on LoRa based Low Power Wide Area Networks (LPWANs). In *Proc. of IEEE WTS*.

[51] TinyOS Alliance. [n. d.]. https://github.com/tinyos/tinyos-main. ([n. d.]).

[52] Shuai Tong, Jiliang Wang, and Yunhao Liu. 2020. Combating packet collisions using non-stationary signal scaling in LPWANs. In *Proc. of ACM MobiSys*.

[53] Transmission of IPv6 Packets over IEEE 802.15.4 Networks. 2020. https://tools.ietf.org/html/rfc4944. (2020).

[54] W3C WoT (Web of Things) Working Group. 2021. https://www.w3.org/WoT/. (2021).

[55] Wei Dong, Jie Yu, and Xiaojin Liu. 2015. CARE: Corruption-Aware Retransmission with Adaptive Coding for the Low-Power Wireless. In *Proc. of IEEE ICNP*.

[56] Wi-Fi & LoRaWAN® Deployment Synergies. 2019. https://lora-alliance.org/resource-hub/wi-fi-lorawanr-deployment-synergies. (2019).

[57] Tim Winter, Pascal Thubert, Anders Brandt, Jonathan W Hui, Richard Kelsey, Philip Levis, Kris Pister, Rene Struik, Jean-Philippe Vasseur, Roger K Alexander, et al. 2012. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. *RFC* 6550 (2012), 1–157.

[58] Yun-Chen Li and Mei-Ling Chiang. 2005. LyraNET: a zero-copy TCP/IP protocol stack for embedded operating systems. In *in Proc. of IEEE RTCSA*.