Scylla: Interleaving Multiple IoT Stacks on a Single Radio

Hassan Iqbal
LUMS
Lahore, Pakistan
16060023@lums.edu.pk

Muhammad Hamad Alizai
LUMS
Lahore, Pakistan
hamad.alizai@lums.edu.pk

Ihsan Ayyub Qazi
LUMS
Lahore, Pakistan
ihsan.qazi@lums.edu.pk

Olaf Landsiedel
Kiel University
Kiel, Germany
ol@informatik.uni-kiel.de

Zartash Afzal Uzmi
LUMS
Lahore, Pakistan
zartash@lums.edu.pk

ABSTRACT

IoT deployments often require communication between devices that employ heterogeneous wireless technologies. Traditionally, expensive gateways are used to relay packets between heterogeneous nodes. Recent cross-technology communication offers a low bandwidth alternative, which is only feasible when communication between such nodes is limited to simple binary commands. In contrast, our work capitalizes on the increasing presence of multi-standard radio chips in mainstream IoT devices, to provide a new perspective on how to enable direct communication between heterogeneous nodes. We design Scylla—a software control layer—that allows multiple wireless stacks to coexist on top of a single radio chip, thereby simultaneously offering multiple communication interfaces. Uniquely, Scylla achieves near stack-native performance and requires no changes to the standards.

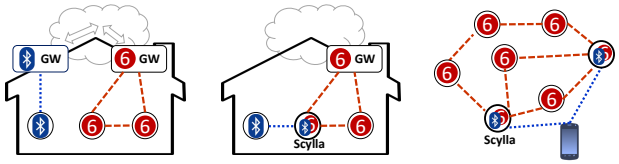
CCS CONCEPTS

• **Networks → Network design principles**; **Network protocol design**; **Network resources allocation**;

ACM Reference Format:
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 *CoNEXT '18: International Conference on emerging Networking EXperiments and Technologies, December 4–7, 2018, Heraklion, Greece.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3281411.3281412

1 INTRODUCTION

Emerging applications in residential, business, automotive, and industrial domains have contributed to an enormous growth of IoT deployments in recent years. These deployments often comprise a wide range of devices, such as sensors, actuators, relays, and compound embedded boards, that communicate based on a variety of wireless technology standards (or *stacks*) such as Zigbee, low-power IPv6 (6LoWPAN), Bluetooth low-energy (BLE), LoRaWAN, Z-Wave, WirelessHART, and IEEE 802.11ah, to name a few.



(a) The gateway approach (b) Scylla as in-network gateway (c) Scylla for edge-computing

Figure 1: Depicting Scylla Core Function. *A traditional gateway approach is shown in (a). A Scylla node in (b,c) provides in-network gateway functionality and facilitates direct communications between heterogeneous devices.*

Many IoT applications in present-day deployments require communication between *heterogeneous* IoT devices (i.e., those using differing communication stacks)[1]. Unfortunately, this is not readily possible today due to their *incompatible* communication interfaces. This is despite the fact that bulk of the IoT communication stacks operate in the unlicensed spectrum, often using even the same frequency bands.

We present Scylla[2], a *software control layer* that enables commodity and inexpensive IoT devices to provide the functionality of a multi-radio gateway. Scylla achieves this by seamlessly *interleaving* multiple wireless stacks on a *single* radio. As a result, heterogeneous IoT nodes are able to communicate at stack-native speeds without requiring any changes in the wireless standards as shown in Fig. 1. Scylla's design is based on two observations:

- The increasing use of multi-standard radio chips, which are rapidly penetrating the IoT device market (cf. Table 1).
- Radio *duty-cycling* support in IoT stacks to conserve energy by avoiding idle listening. This leaves time in between the transmissions from a single stack, allowing co-located stacks to interleave their transmissions as shown in Fig. 2.

[1] A floor supervisor in a manufacturing unit might need to use their smartphone to collect sensor data and exchange control commands with a LoRaWAN-equipped device used for process tracking and control.
[2] A multi-headed monster in Greek mythology.

CoNEXT '18, December 4–7, 2018, Heraklion, Greece

ACM ISBN 978-1-4503-6080-7/18/12…$15.00
https://doi.org/10.1145/3281411.3281412

Table 1: Multi-standard radio chips and platforms

Chip(s)	Platform	BLE	6LoWPAN	Zigbee	Wifi	LoRA	SigFox	NFC
CC2650	TI Sensortag[10]	✓	✓	✓	-	-	-	-
CC1350	TI Sensortag[11]	✓	-	-	-	✓	-	-
nRF52840	NS nRF52840 DK[25]	✓	✓	✓	-	-	-	-
K32W042	NXP KW32W0x WP[22]	✓	✓	✓	-	-	-	-
STM32WB55	STM Nucleo[27]	✓	✓	✓	-	-	-	-
BCM43438	Redbear Duo[23]	✓	-	-	✓	-	-	-
nRF52832	Arduino Primo[3]	✓	-	-	✓	-	-	✓
NINA-W10	Arduino Vidor[4]	✓	-	-	✓	-	-	-
BCM2837B0	RaspbarryPi 3B[20]	✓	-	-	✓	-	-	-
BCM2835	RaspberryPi ZeroWH[21]	✓	-	-	✓	-	-	-
ML7404[24]	LAPIS Semiconductor	-	✓	✓	-	-	✓	-
WE866C3[28]	Telit	✓	-	-	✓	-	-	-

A Scylla node is *functionally* similar to a multi-radio gateway, which also provisions multiple communication stacks. However, Scylla offers a lower cost alternative by using a commodity IoT device that employs a single radio interface[3]. As a result, Scylla obviates the need for complex coordination mechanisms—required in a multi-radio gateway to avoid packets collisions—by consolidating the the stack coordination function within the Scylla layer. Compared to such multi-radio gateways, Scylla does not incur any additional energy overhead, as its radio duty-cycle is simply an aggregation of the duty-cycles of the interleaved stacks.

Nonetheless, realizing Scylla is challenging. First, wireless stacks usually have their activity schedules. In the absence of global coordination, transmission overlap (i.e., conflict) between interleaving stacks can degrade performance. Second, existing system support for multi-standard radio chips only offers *static* configuration of a single wireless stack at any time. Scylla addresses the first challenge through dynamic priority-based interleaving, which prevents starvation in heavy traffic conditions and allows weighted fair sharing across stacks. We address the second challenge by keeping separate data structures and by dynamically rewriting interfaces between layers using function pointers.

Besides the basic gateway configuration, Scylla supports a number of other infrastructure configurations and use-cases. These include provisioning of a low-latency edge computing platform via BLE smartphones in a 6LoWPAN mesh (see Fig. 1(c)) [6], improving data fidelity in crowd sensing [30] by augmenting smartphone data with streams from nearby sensors, offering data muling services in remote areas [19, 29], and enabling network administrators to directly and securely spread commands or collect data. Furthermore, Scylla may also be feasible for peer-to-peer communication between sensors in easily accessible deployments with good energy provisioning, such as smart homes.

Scylla is not the first to explore cross-technology communication (CTC) of IoT devices. Previous proposals include "PHY emulation [5, 12, 14]" or "generating universally detectable energy patterns [9, 16]". These proposals, however, are useful only for sending command and control messages with insignificant amounts of low-rate data (such as controlling a Zigbee bulb through a BLE smartphone [12]) as they only provision a low-bandwidth communication channel.

We implement Scylla in Contiki OS and evaluate it on TI Sensortag platform [10] using raw data transfers as well as CoAP, a widely used lightweight application protocol for IoT devices. Our experiments demonstrate that Scylla sustains only a negligible performance penalty—mostly between 0-5% depending upon traffic load—in key metrics such as packet delivery, energy efficiency, throughput, latency, and end-to-end transport reliability, while remaining standards-compliant. This is important for system integrators to keep the cost and complexity low when commissioning a deployment of heterogeneous IoT devices.

[3]It is not uncommon for traditional IoT gateways to cost upwards of 150 USD [14]; this amount may be sufficient to buy a dozen or more IoT devices.

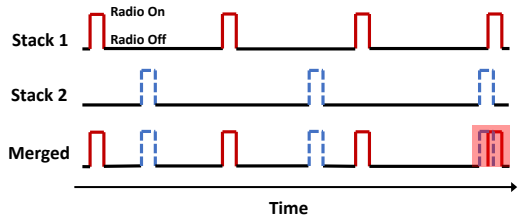


Figure 2: Scylla exploits radio duty-cycling in IoT devices to overlay multiple stacks over a single radio. *The figure shows merged transmissions from two stacks; a possible overlap is addressed by Scylla's design detailed in Sec. 2.*

We make the following contributions in this paper:

(1) Design of Scylla, which allows different technology stacks to simultaneously operate over a single radio without requiring any protocol modifications or inter-stack coordination.
(2) A pilot implementation of Scylla for interleaving two example stacks (BLE and 6LoWPAN) while meeting the efficiency design goals.
(3) A detailed evaluation using real-world IoT devices to demonstrate that Scylla nodes successfully expose both the interfaces to the external world while supporting operations at near stack-native rates.

We open source Scylla (https://github.com/iqbal-h/contiki/) to enable heterogeneous deployments without traditional gateways, and to facilitate research on interleaving other wireless technologies.

2 DESIGN DESCRIPTION

The design of Scylla aims to overcome the challenges highlighted in Sec. 1, namely: (i) conflicts in radio transmissions from different stacks in the absence of global coordination, and (ii) lack of IoT device resources to support *dynamic* switching between the stacks. Our design attempts to address these challenges with the following as a complete list of design goals:

2.1 Design Goals

- **G1: Low Overhead.** The design should impose low overhead (less energy or wasted processing time) when dynamically interleaving wireless stacks over a single radio.
- **G2: High Performance.** Given typical radio duty-cycling in IoT devices, the design should support stack-native packet delivery, throughput, and latency performance.
- **G3: Flexibility.** The proposed scheme should be able to work in a heterogeneous environment, where the number of devices with multi-stack support is limited.

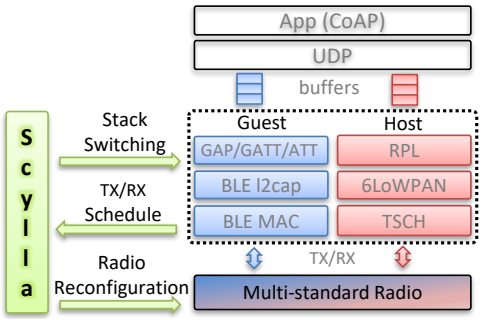


Figure 3: Scylla implementation architecture.

- **G4: Compatibility.** Our design should not require any changes in the wireless standards being interleaved.

2.2 Stack Dependency of Design

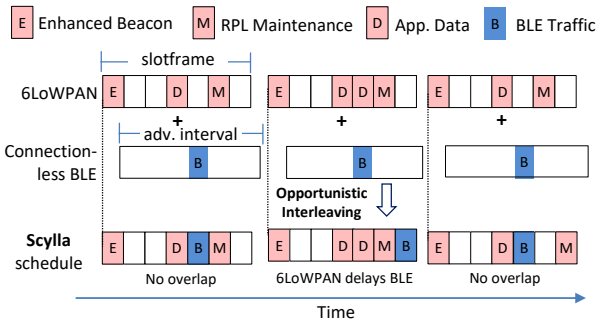
Scylla interleaves transmissions from multiple communication stacks; its design may be optimized by considering the underlying medium access control (MAC) technique for each stack. However, we take a generic approach and design Scylla to work with *all popular* time-synchronized link layers (BLE, Z-Wave, WirelessHART, and 6LoWPAN, etc.). These layers follow their own strict, globally-defined, activity schedules and may result in colliding transmissions (see Fig. 2). Stacks using asynchronous MACs (e.g., S-MAC [31]) work with short duty cycles, typically below 5%, and thus may allow simple round-robin interleaving of transmissions without any inter-stack collisions.

2.3 Prototype implementation

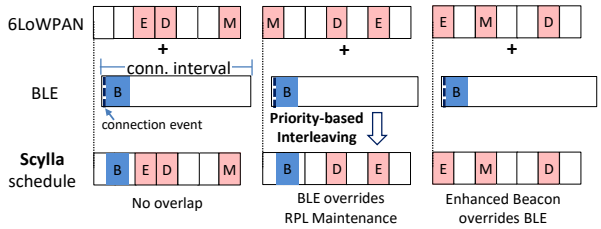
Our prototype implementation follows the generic design of Scylla and targets BLE and 6LoWPAN as the interleaving stacks. Fig. 3 depicts an architectural view of the implementation; we regard the 6LoWPAN as the *host* stack indicating that its transmission schedule is considered by Scylla as a reference time-line to steer the interleaving process. The BLE acts as the *guest* stack such that its transmissions are interleaved into those of 6LoWPAN.

Host Stack. We employ RPL routing [1] along with 6LoWPAN [17] at the network layer. At the MAC layer, we use IEEE 802.15.4e (Time-slotted Channel Hopping - TSCH) [8] which groups transmission slots into *slotframes*. To conserve energy, one node is allocated a single time slot, typically of 10ms duration (sufficient for making a transmission and getting an ack with encryption), within one *slotframe*. We use Orchestra [7] scheduler to determine if a node should transmit, receive, or sleep during an allocated time slot.

Guest Stack (BLE). The BLE protocol suite supports both connection-less and connection-based communication modes. We use both these modes in our prototype. In the connection-less mode, contention between broadcast *advertisements*—sent at configurable intervals by low-power IoT devices to nearby portable-electronic devices (Tablets, Smartphones, etc.)—is resolved through random delays. In contrast, the connection-based mode imposes periodic communication (and synchronization) between a *master* and its



(a) Opportunistic Interleaving



(b) Priority-based Interleaving

Figure 4: Interleaving approaches. *(a) Opportunistic approach interleaves connection-less BLE during free 6LoWPAN time slots. (b) Priority-based approach defines per-packet priority to resolve overlapping time slots between connection-based BLE and 6LoWPAN.*

slaves at *connection events* occurring after a fixed *connection interval*; the master may specify the number of connection events that a slave may ignore without connection reset.

2.4 Interleaving Stacks

Scylla offers two different interleaving mechanisms:

2.4.1 Opportunistic Interleaving. In this setup, Scylla sticks to the transmission schedule of the time-synchronized *host* stack—6LoWPAN in our prototype design. Transmissions from a contention-based *guest* stack are then interleaved onto the next available empty time slots. Fig. 4(a) depicts this with three consecutive *slotframes* example. In the first *slotframe* (left), transmission schedules from the two stacks do not conflict and Scylla simply superimposes them. The transmission from the *guest* stack in the middle *slotframe* is delayed by Scylla to avoid a conflict. This is acceptable in the standard. Finally, the *guest* transmission returns to the original schedule in the last *slotframe* (right). This interleaving scheme achieves our goals **G3** and **G4**.

2.4.2 Priority-based Interleaving. If the *guest* stack is also time-synchronized, its scheduled transmissions conflicting with those from the *host* stack may not be delayed[4]. To resolve conflicts, then, we prioritize transmissions—either with static priorities or with per-transmission priorities computed dynamically. Fig. 4(b) illustrates

[4]For connection-based BLE, this is because the *master* and the *slaves* can only communicate at fixed intervals.

Table 2: Scylla implementation vs memory sizes of IoT devices with multi-standard radio chips

Platform	Flash (kB)	RAM (kB)	Scylla fits?
CC2650 Sensortag[10]	128	20	✓
CC1350 Sensortag[11]	128	20	✓
nRF52840 DK [25]	$1x10^3$	256	✓
NXP KW32W0x WP[22]	$1.25x10^3$	384	✓
Nucleo[27]	$1x10^3$	64	✓
Redbear Duo	$1x10^3$	128	✓
Arduino Primo[3]	512	64	✓
Arduino Vidor[4]	256	32	✓
Raspbarry Pi 3 B+ [20]	External	$1x10^6$	✓
Raspberry Pi Zero WH[21]	External	$512x10^3$	✓

the priority-based interleaving, which handles the overlapping radio schedules of individual stacks based on priority of different transmissions.

Static Priorities. In this case, Scylla schedules and preempts transmissions such that *enhanced beacons* take the highest priority, followed by BLE traffic (connection events), and then 6LoWPAN *application data* and *routing maintenance*. This priority order is chosen because: (i) beacons are critical for network-wide synchronization and have the highest priority in TSCH [7], (ii) app data, if unacknowledged, gets re-transmitted anyways by RPL [1], and (iii) occasional failure of routing maintenance signals is arguably less critical than missing the BLE connection event between master and slave. Moreover, Scylla avoids conflicts with repetitive patterns by using TSCH *slotframe* size and BLE connection intervals to be mutually prime.

Dynamic Priorities. The static scheme does not require any complex processing (meeting goal **G1**). It works well in sparse traffic conditions but may starve transmissions from the low-priority stack (6LoWPAN, in our case) under heavy traffic (Sec. 3). To handle deployments with high traffic load, Scylla uses a dynamic priority scheme: it flips a coin to randomly choose the priority in each time slot, whenever a conflict arises. By changing the bias of the coin, Scylla can achieve a variety of sharing disciplines (weighted or fair) to appropriately handle varying traffic loads on the stacks.

With these interleaving mechanisms, a Scylla node can assume the role of either a master or a slave in a BLE network. New connections can be established opportunistically using empty slots of the *host* stack, followed by priority-based interleaving during the lifetime of a connection.

2.5 Implementation Challenges

Our Scylla prototype uses the native implementations of BLE [26] and 6LoWPAN [17] stacks in Contiki [18] for TI SensorTag platform [10]. The entire implementation (interleaving logic + BLE + 6LoWPAN) in Contiki consumes 92.16 kB in flash (code memory) and 18.13 kB in RAM (data memory). This should comfortably fit into most commodity IoT platforms with similar multi-standard radio chips, as illustrated in Table 2. Practically, Scylla needs the ability to dynamically (i) reconfigure the radio, and (ii) switch between the communication stacks.

Radio Reconfiguration. This is simply done by restarting the radio chip with appropriate parameters. We accomplish this by building a wrapper around radio on/off routines to redirect radio control towards Scylla. At the start of every time slot of the *host* stack, Scylla determines the standard that the radio must be reconfigured with, depending on the interleaving logic described above and whether a node has to transmit, receive, or continue to sleep. As a result, the process does not consume any more time and energy than is needed to reboot a sleeping radio using its previous configuration (meeting the design goal **G1**).

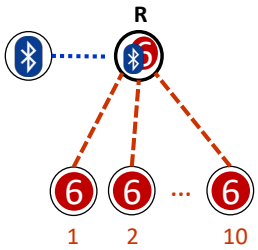
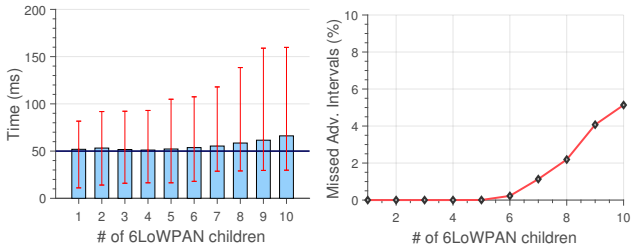


Figure 5: Evaluation topology. *R is a 6LoWPAN root that runs Scylla to communicate with a BLE node. We vary the number of 6LoWPAN children across experiments.*



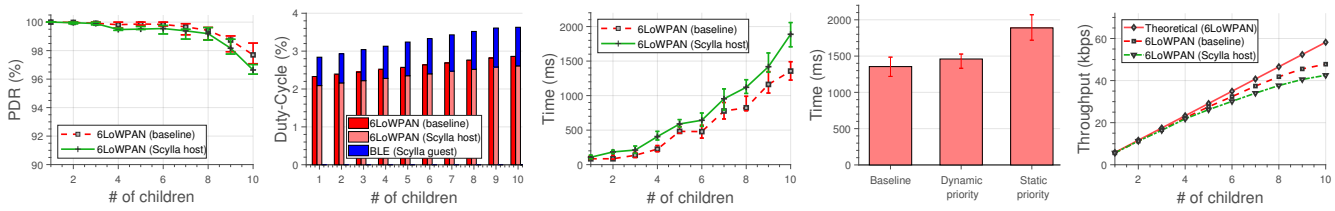
(a) BLE latency (b) Missed Advertisements

Figure 6: Guest stack latency with Opportunistic Interleaving: *(a) the avg. latency is at par with the baseline (horizontal line at 50ms), and (b) the number of missed advertisement interval stays below 5% even at heavy traffic.*

Stack Switching. Switching network stacks is tricky as IoT OSes rarely provide dynamic switching support. For example, Contiki uses compile time macros (i.e., a reference to the binary file) to define a protocol that should be installed at a particular layer of the networking stack. We update the Contiki architecture to enable this support at runtime. This consists of defining relevant data structures that can maintain the configuration of each network stack, as well as function pointers to dynamically rewrite the interfaces between layers. Scylla uses these data structures to activate or freeze a stack at a given time depending on the radio configuration being applied.

3 EVALUATION

Our evaluation of Scylla centers around two key questions: (i) *what impact does Scylla bear on the performance of an interleaved stack?* and (ii) *what application-level benefits does Scylla provide?* For answering the first question, we consider raw data transfers whereas the latter is ascertained by deploying CoAP (a popular application protocol for IoT devices) on heterogeneous IoT nodes



(a) PDR (b) Energy (c) Latency (d) Latency with dyn. priorities (e) Throughput

Figure 7: 6LoWPAN performance as Scylla *host* stack. *A small penalty in (a) PDR and (b) Energy relative to the baseline case (i.e., no Scylla). The undesirable latency with* static *priority in (c) may be adequately addressed using dynamic priority schedule in (d). A small throughput loss in (e) increases with the number of 6LoWPAN children nodes.*

interconnected via a Scylla-enabled relay. We first describe our experimental setup.

3.1 Experimental Setup and Metrics

We use a 6LoWPAN network, shown in Fig. 5, with one root node R and a number of children, which we vary from 1 to 10. We use the following metrics, measured at node R: (i) packet delivery ratio (PDR), (ii) energy, (iii) latency, and (iv) throughput. The results are obtained when R operates with and without Scylla. The latter forms our baseline.

Raw data transfer. These experiments additionally employ a single BLE node along with the 6LoWPAN network (see Fig. 5). It may be noted that a single BLE connection can be used to mimic multiple BLE connections by using a smaller *connection interval*. When R is enabled with Scylla, traffic generated by 6LoWPAN children is aggregated by R and forwarded onto the BLE link. Of course, when R does not run Scylla, the BLE node is isolated and R sinks the traffic generated on the 6LoWPAN network.

CoAP communication. These experiments also use a single BLE node running a CoAP server. The server maintains bidirectional communication with CoAP clients running on 6LoWPAN children. The Scylla-enabled root node R aggregates and relays the CoAP traffic in each direction. When R does not run Scylla, the BLE node is isolated; in that case, we use one of the 6LoWPAN children nodes to run the CoAP server and communicate via R with CoAP clients running on other 6LoWPAN children.

3.2 Establishing Feasibility of Scylla

Scylla's impact on an interleaved stack stems from how it handles overlapping activity schedules which, in turn, depends on the underlying interleaving mechanism.

3.2.1 Opportunistic Interleaving. In this case, Scylla does not impact the *host* stack. However, the opportunistic operation impacts the transmission timing and the performance of the *guest* stack. The results in Fig. 6, obtained over an interval spanning 10,000 BLE transmissions, demonstrate that this impact is minimal: the average latency (thick bars in Fig. 6(a)) remains at par with the baseline (the configured 50ms interval depicted by horizontal line) and even for 10 children nodes, the average latency increases by only 15ms, which is acceptable in typical IoT applications based on connection-less BLE [26]; and the number of intervals in which no advertisements could be transmitted due to unavailability of free time slots stays below 5% even with dense 6LoWPAN neighborhood (see Fig. 6(b)). It may be noted that the error-bars in Fig 6(a) do not represent the confidence intervals; rather, they depict the best and worst data-point of the entire experimental run.

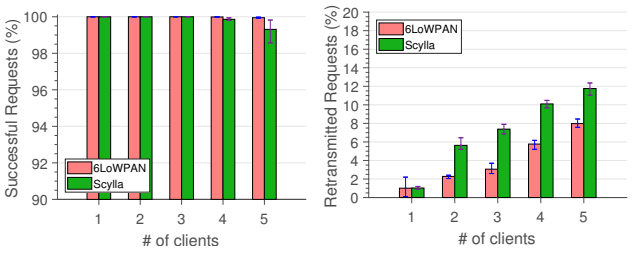
3.2.2 Priority-based Interleaving. In this case, the *guest* stack (a *connection-based* BLE) may impact the performance of *host* stack (6LoWPAN) significantly. To quantify the worst case performance for 6LoWPAN, we provide results for the *static* priority-based interleaving mechanism, which prioritizes BLE traffic over 6LoWPAN data traffic. We set the BLE connection interval to 210ms for stress-testing, and use the default settings of TSCH/6LoWPAN in Contiki including the *slotframe* size of 170ms. The metrics of interest for 6LoWPAN are depicted in Fig. 7(a) through Fig. 7(c) by taking an average of three repeated runs of an experiment, each lasting 30 min.

PDR. Fig. 7(a) indicates a small drop (<4%) in PDR with increasing number of 6LoWPAN children. This is expected because with larger neighborhood, a BLE transmission is more likely to conflict with any 6LoWPAN child, causing a drop in PDR. The drop, however, remains negligible.

Energy. Fig. 7(b) shows that Scylla operates well below 5% duty-cycle. Compared to the baseline, the cumulative transmission (*host* + *guest*) is slightly higher, showing an upward trend with the number of 6LoWPAN children. Surprisingly, the duty cycle of *host* stack (6LoWPAN) is lower than the baseline; this is because of the subtraction of those overlapping slots that get assigned to BLE.

Latency. Fig. 7(c) indicates that Scylla has a noticeable impact on the latency of 6LoWPAN. This is because of low *static* priority of 6LoWPAN data which, when preempted by the BLE, needs retransmission. The latency impact increases with the number of 6LoWPAN children and may be critical for certain latency-sensitive IoT applications (e.g., M2M [15] and Industry 4.0 [13]). We address this using *dynamic* priorities that allow 6LoWPAN to occupy more conflicting slots. Fig. 7(d) confirms that this is indeed the case; the additional latency introduced by Scylla drops from over 30% to ≈5% in a 10-children configuration.

Throughput. To compute the throughput, we configure 6LoWPAN root node R in Fig. 5 to send data at full rate, in a round-robin



(a) Request success rate (b) App. level retransmissions

Figure 8: CoAP over Scylla. *The success rate of CoAP requests is comparable to the baseline case (without Scylla) while causing a slight increase in application level retransmissions.*

manner, to each child. As the number of children increases, causing a growth in control traffic, the baseline throughput is expected to fall below the theoretical max (see Fig. 7(e)). When node R is enabled with Scylla, it also sends data on the BLE link in each connection event, causing a further decrease in 6LoWPAN (*host* stack) throughput.

Altogether, we conclude that the impact of Scylla on the performance of an interleaved stack is negligible—meeting our design goal **G2**. Scylla's latency control feature, realized through *dynamic* priority scheduling, keeps delays low even at high loads.

3.3 Application-level Benefits of Scylla

We now present results from our CoAP deployment. While CoAP is designed to interface with HTTP, it provides a connection-less request/response interaction model between application endpoints (clients and servers) for exchanging sensor values. In this evaluation, each CoAP client issues back-to-back requests to the server at the fastest possible rate. Simultaneous requests from a single client are not supported in the Contiki's implementation of CoAP, as it is designed for low-rate transport.

Fig. 8 shows (a) CoAP request success rate, and (b) CoAP level retransmissions, with up to 5 CoAP clients[5], each sending 5000 requests. Similar to the baseline, Scylla-enabled heterogeneous network retains a very high (i.e., >99%) request success rate. However, Scylla causes a higher number of app level retransmissions, increasing with the number of 6LoWPAN children. This is due to two reasons: (i) an increase in the number of overlapping slots, and (ii) lack of link-layer retransmission support in, to the best of our knowledge, the only open-source implementation of BLE [26].
Summarizing, our raw data transfer and CoAP-based results indicate that Scylla is not just a feasible in-network solution for heterogeneous IoT deployments, it is also efficient based on commonly used performance criteria for IoT deployments.

4 RELATED WORK AND DISCUSSION

Scylla enables direct communication between heterogeneous IoT nodes without using multi-radio gateways [2], which are costly, induce longer routing paths, and are typically a burden for unplanned IoT deployments [14].

[5]CoAP server ran out of memory beyond 5 clients.

Cross-technology communication (or CTC) [5, 12, 14, 16] offers initial insights on how to reduce reliance on gateways using legacy, uni-standard radio chips. CTC techniques, which can broadly be divided into two classes, exploit the opportunities exposed by the co-existence of wireless technologies in the ISM band.

Pulse position modulation (PPM) based CTC techniques [9, 14, 32] modulate symbols by shifting the transmission timing of packets, such as broadcast beacons. These shifts in timing are sensed and the corresponding symbols are interpreted by the receiver through repeated RSSI sampling. Although this method requires no additional bandwidth or energy, it is highly constrained in terms of throughput - as low as a few symbols per second.

PHY emulation [12, 16] based CTC techniques smartly select payload bits in one wireless standard to emulate complete packets of another. Implementations exist for WiFi→Zigbee [16], BLE→WiFi [5] and BLE→Zigbee [12] packet exchanges. Although this technique improves throughput by orders of magnitude compared to PPM based approaches, its utility for full data transfers is still not established due to the high emulation overhead.

In contrast, Scylla leverages the increasing presence of multi-standard radio chips to offer data transfer at near stack-native speeds across a network of heterogeneous nodes, as demonstrated by CoAP deployment in Sec. 3.

Our work, nonetheless, has limitations. The empirical evidence we provide is focused on just two stacks, 6LoWPAN and BLE; despite their popularity in IoT, they still represent a single point in the vast landscape. Similarly, the deployment of CoAP over a Scylla-enabled relay represents a pragmatic, yet single example. However, we believe that our work can serve as a stepping stone to understand challenges in interleaving multiple wireless stacks on IoT devices. For example, the interleaving strategies we propose are likely applicable to other wireless technologies, and can be used to support more than two stacks. Similarly, we showed that the energy and memory overhead of Scylla is commensurate with the number of interleaved stacks. Hence, for example, interleaving more than two stacks on TI SensorTag is not possible but may still be feasible in a number of other IoT device platforms that can bear the corresponding overhead, as shown in Table 2.

5 CONCLUSION

We presented Scylla, a software control layer that allows multiple wireless stacks to coexist on top of a single, multi-standard radio chip. The net result is a ubiquitous communication support in heterogeneous environments at a reasonable memory cost (i.e., easily fits in the memory of commodity platforms) and energy overhead (i.e., <5% radio duty-cycle). Our evaluation, based on 6LoWPAN as *host* and BLE as *guest*, provides empirical evidence that Scylla is able to expose both interfaces to the external world without noticeable performance degradation in key metrics such as packet delivery, energy, throughput, latency, and reliable transport.

ACKNOWLEDGEMENTS

We are thankful to the anonymous reviewers and our shepherd, Swarun Kumar, for the insightful comments. This research is in part funded by the Higher Education Commission (HEC) of Pakistan through grant NRPU-4147.

REFERENCES

[1] Roger Alexander, Anders Brandt, JP Vasseur, Jonathan Hui, Kris Pister, Pascal Thubert, P Levis, Rene Struik, Richard Kelsey, and Tim Winter. 2012. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550. (March 2012).

[2] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio. 2016. A Mobile Multi-Technology Gateway to Enable IoT Interoperability. In *First IEEE International Conference on Internet-of-Things Design and Implementation, IoTDI 2016, Berlin, Germany, April 4-8, 2016*. 259–264.

[3] Arduino. 2017. *Arduino Primo*. https://store.arduino.cc/.

[4] Arduino. 2018. *Arduino Vidor*. https://store.arduino.cc/.

[5] Zicheng Chi, Yan Li, Hongyu Sun, Yao Yao, Zheng Lu, and Ting Zhu. 2016. B2W2: N-Way Concurrent Communication for IoT Devices. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM (SenSys '16)*. ACM, New York, NY, USA, 245–258.

[6] Junguk Cho, Karthikeyan Sundaresan, Rajesh Mahindra, Jacobus Van der Merwe, and Sampath Rangarajan. 2016. ACACIA: Context-aware Edge Computing for Continuous Interactive Applications over Mobile Networks. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. ACM, New York, NY, USA, 375–389.

[7] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. 2015. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15)*. ACM, New York, NY, USA, 337–350.

[8] IEEE Standard for Low-Rate Wireless Networks Std 802.15.4-2015. 2016. (April 2016).

[9] X. Guo, X. Zheng, and Y. He. 2017. WiZig: Cross-technology energy communication over a noisy channel. In *2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, May 1-4, 2017*. 1–9.

[10] Texas Instruments. 2016. *CC2650 SensorTags*. http://www.ti.com/.

[11] Texas Instruments. 2017. *CC1350 SensorTags*. http://www.ti.com/.

[12] Wenchao Jiang, Zhimeng Yin, Ruofeng Liu, Zhijun Li, Song Min Kim, and Tian He. 2017. BlueBee: A 10,000x Faster Cross-Technology Communication via PHY Emulation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys '17)*. ACM, New York, NY, USA, Article 3, 13 pages.

[13] Anders Ellersgaard Kalør, René Guillaume, Jimmy Jessen Nielsen, Andreas Mueller, and Petar Popovski. 2017. Network Slicing for Ultra-Reliable Low Latency Communication in Industry 4.0 Scenarios. *CoRR* abs/1708.09132 (2017).

[14] Song Min Kim and Tian He. 2015. FreeBee: Cross-technology Communication via Free Side-channel. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*. ACM, New York, NY, USA, 317–330.

[15] Akshay Kumar, Ahmed Abdel-Hadi, and T. Charles Clancy. 2016. An online delay efficient packet scheduler for M2M traffic in industrial automation. In *Annual IEEE Systems Conference, SysCon 2016, Orlando, FL, USA, April 18-21, 2016*.

[16] Zhijun Li and Tian He. 2017. WEBee: Physical-Layer Cross-Technology Communication via Emulation. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. ACM, New York, NY, USA, 2–14.

[17] Gabriel Montenegro, Christian Schumacher, and Nandakishore Kushalnagar. 2007. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 4919. (Aug. 2007).

[18] Contiki OS. 2004. *Contiki: The Open Source OS for the Internet of Things*. http://www.contiki-os.org/.

[19] Unkyu Park and John Heidemann. 2011. Data Muling with Mobile Phones for Sensornets. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*. ACM, New York, NY, USA, 162–175.

[20] Pi3Bplus. 2018. *RaspbarryPi*. https://www.raspberrypi.org/magpi/raspberry-pi-3b-plus/ (accessed 2018-05-23).

[21] PiZeroWH. 2018. *RaspbarryPi*. https://www.adafruit.com/product/3708 (accessed 2018-05-23).

[22] NXP 32Wx Platform. 2018. *NXP*. https://www.nxp.com/.

[23] Redbear. 2016. *Redbear Duo*. https://redbear.cc/product/wifi-ble/redbear-duo.html (accessed 2018-05-23).

[24] Lapis Semiconductor. 2017. *ML7404*. http://www.lapis-semi.com/.

[25] Nordic Semiconductor. 2018. *Nodric Dev Kit*. https://www.nordicsemi.com/.

[26] 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 *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys '17)*. ACM, New York, NY, USA, Article 2, 14 pages.

[27] STMicroelectronics. 2018. *Nucleo*. http://www.st.com/.

[28] Telit. 2018. *WE866C3*. https://www.telit.com/wp-content/uploads/2018/03/Telit_WE866C3_Datasheet-1.pdf (accessed 2018-05-23).

[29] Deepak Vasisht, Zerina Kapetanovic, Jong-ho Won, Xinxin Jin, Ranveer Chandra, Ashish Kapoor, Sudipta N. Sinha, Madhusudhan Sudarshan, and Sean Stratman. 2017. Farmbeats: An IoT Platform for Data-driven Agriculture. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*. USENIX Association, Berkeley, CA, USA, 515–528.

[30] Yu Xiao, Pieter Simoens, Padmanabhan Pillai, Kiryong Ha, and Mahadev Satyanarayanan. 2013. Lowering the Barriers to Large-scale Mobile Crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications (HotMobile '13)*. ACM, New York, NY, USA, Article 9, 6 pages.

[31] Wei Ye, J. Heidemann, and D. Estrin. 2002. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, New York, USA,*, Vol. 3. 1567–1576 vol.3.

[32] S. Yin, Q. Li, and O. Gnawali. 2015. Interconnecting WiFi Devices with IEEE 802.15.4 Devices without Using a Gateway. In *2015 International Conference on Distributed Computing in Sensor Systems, DCOSS 2015, Fortaleza, Brazil, June 10-12, 2015*. 127–136.