



Indriya2: A Heterogeneous Wireless Sensor Network (WSN) Testbed

Paramasiven Appavoo^(✉), Ebram Kamal William, Mun Choon Chan,
and Mobashir Mohammad

National University of Singapore, Singapore, Singapore
{pappavoo, ebramkw, chanmc, mobashir}@comp.nus.edu.sg

Abstract. Wireless sensor network testbeds are important elements of sensor network/IoT research. The Indriya testbed has been serving the sensor network community for the last 8 years. Researchers from more than a hundred institutions around the world have been actively using the testbed in their work. However, given that Indriya has been deployed for over 8 years, it has a number of limitations. For example, it lacks support for heterogeneous devices and the ability to handle data generated by the testbed with no loss, even at a relatively low sampling rate. In this paper, we present the design and implementation of an upgraded version of Indriya, Indriya2, with the following improvements, namely (1) support for heterogeneous sensor devices, (2) support for higher data rate through the infrastructure, (3) support for multiple users to schedule jobs over non-overlapping set of heterogeneous nodes at the same time, and (4) a real-time publish/subscribe architecture to send/receive data to/from the testbed nodes.

Keywords: Testbed · Internet of Things · Wireless sensor network

1 Introduction

With the emergence of Internet-of-Things (IoT), the ability to experiment and evaluate different sensor network protocols in large, realistic environments continued to be important. Indriya [1], is a wireless sensor network testbed deployed at the School of Computing, National University of Singapore. Indriya has been available as an open testbed for use by the research communities for more than 8 years (since December 2009). It has served more than 500 users from over 150 institutions with over 13,000 jobs executed.

While Indriya has served its purpose well, it has limitations. First, its software is derived from MoteLab [18], an even older wireless sensor network testbed and much of the internal design is based on software tools and components that are very dated. This makes system upgrade and maintenance difficult. Second, the current design of Indriya can support only one single hardware platform (TelosB [4]). With many new sensor hardware platforms available in the last few years

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2019

Published by Springer Nature Switzerland AG 2019. All Rights Reserved

H. Gao et al. (Eds.): TridentCom 2018, LNCS 11270, pp. 3–19, 2019.

https://doi.org/10.1007/978-3-030-12971-2_1

and with more likely to emerge in the future, it is important that the testbed is able to support newer hardware platforms.

In this paper, we present the design of Indriya2. Indriya2 is designed to address the limitations of Indriya and has the following objectives. First, it should be able to support different sensor platforms and wireless network technologies. Second, the infrastructure should support higher data rate to handle high influx of time-stamped data generated by nodes for either debugging or application purposes. The need to support higher aggregated data rate comes from the possible increase in either number of sensor nodes in the testbed or higher traffic rate generated to support experimentation.

We have completed the design and implementation of Indriya2. The main features are summarized as follow:

1. It can support different hardware platforms. Currently, Indriya2 can support a mixture of TelosB, SensorTag CC2650 and SensorTag CC1350. New device types can be easily added to the testbed by adding a small device specific components for flashing. Indriya2 also supports different operating systems (TinyOS [19] and Contiki [20]) and wireless network technologies (BLE and IEEE 802.15.4g).
2. The internal design of Indriya2 has been completely overhauled to improve software portability and overall performance. It uses a time-series database, InfluxDB, that supports high influx of streaming sensor data. Experiments show that the database can easily support over 2000 transactions per second running on a mid-range commodity server.
3. The front-end interface allows multiple users to run multiple experiments at the same time using different set of nodes. For real-time sensor data monitoring, all the data generated by the active nodes, i.e. under running experiments, are available in real-time through a MQTT server.

The paper is organized as follows. We briefly presented related work in Sect. 2 and recap the design of Indriya in Sect. 3. The design and user interface of Indriya2 are covered in Sect. 4 and 5 respectively. Testbed results are presented in Sect. 6 and conclusion in Sect. 7.

2 Related Work

In this section, we will give a short description of the other common testbeds for wireless sensor networking that are popular in the research community.

FlockLab [14] is a wireless sensor network testbed developed and run by the Computer Engineering and Networks Laboratory at the Swiss Federal Institute of Technology Zurich in Switzerland with TinyNode184, Tmote Sky (TelosB), Opal, OpenMote, and MSP430-CCRF as the sensor motes. It provides reliable power and energy measurements while JamLab [15] allows for generation and reproduction of interference patterns.

FIT IoT-LAB [16] is a large scale infrastructure for wireless sensor networks developed by a consortium of five French institutions of higher education and

research with over 2000 nodes using WSN430, M3, and A8 as the sensor platform. One unique feature of the IoT-LAB is that it includes mobile robots.

The TKN WIreless NetworkS Testbed (TWIST) [17] is developed by the Telecommunication Networks Group (TKN) at the Technische Universität Berlin. TWIST uses 46 single-board and wall-powered computers to manage 204 testbed nodes. Sensor nodes include both the TmoteSky and eyesIFXv2 nodes.

Indriya2 mostly uses the same USB wiring and hardware infrastructure as Indriya. The major changes are in the software design and architecture which allows more support for requirements of an Internet of Things platform, including heterogeneous hardware that supports different wireless technology, and timestamped data access through MQTT service.

3 Indriya

Indriya is deployed across three different floors of our main School of Computing building (COM1). Indriya2's deployment remains the same. The deployment covers spaces used for different purposes, including laboratories, tutorial rooms, seminar rooms, study areas, and walkways. The backend is based on a cluster-based design with each cluster consisting of a single cluster-head. The motes are connected to the cluster-head using USB hubs and active USB cables. The cluster-heads are connected to a server via Ethernet. The server manages the testbed and provides a user interface.

Over the past 8 years, Indriya has registered more than 500 users from more than 150 institutions and over 25 countries. More than 65% of the users are from overseas, including India, Germany, Switzerland, Sweden, United Kingdom and Brazil. The total number of jobs scheduled exceeds 13,000 averaging more than 1,400 per year.

Due to the deployment inside a very active university building, the devices experience real world interference from coexisting WiFi operating in the 2.4 GHz spectrum. This give the users a very realistic testbed to try out and improve their protocols. Besides interference, due to the 3 dimensional deployment of nodes across the three levels of the building, the communication suffers from random occlusion from different objects, giving constantly changing topologies of up to 7 hops at maximum transmission power.

In spite of its popular usage, there are a number of limitations to Indriya's design and implementation. We highlight some of the issues below.

- The design is tightly coupled with TinyOS [2] infrastructure, whose latest release is dated August 2012. Additional class files are required to properly format the output.
- The MySQL database used is not suited for high influx of time series data produced by the sensor network due to the combination of large number of sensor motes sending data on the back channel at (relatively) high rate.

- The system supports only one type of node, that is the CrossBow TelosB [4]. TelosB is a very old platform with limited sensors. The sensor data collected is also relatively noisy when compared to newer sensor hardware like the CC2650 SensorTag [3].
- The source is written in multiple programming languages including C, Perl, and PHP with numerous dependencies, making it difficult to maintain and update.

Indriya2 is designed to address these issues. We will present its design in the next two sections.

4 System Architecture

The system architecture for Indriya2 is shown in Fig. 1. The main components are (1) nodes, like TelosB, SensorTags cc2650, and others, (2) gateways, which are mini desktops like mac-mini and tiny-lenovo, and (3) server(s). Currently, the gateways for Indriya2 are 6 mac-minis running Ubuntu 14 and the server is a quad-core machine with 16 GB RAM running Ubuntu 16.

4.1 Hardware

In this section, we will show how Indriya2 is deployed at the School of Computing of National University of Singapore. Figure 2 shows a picture of cc2650

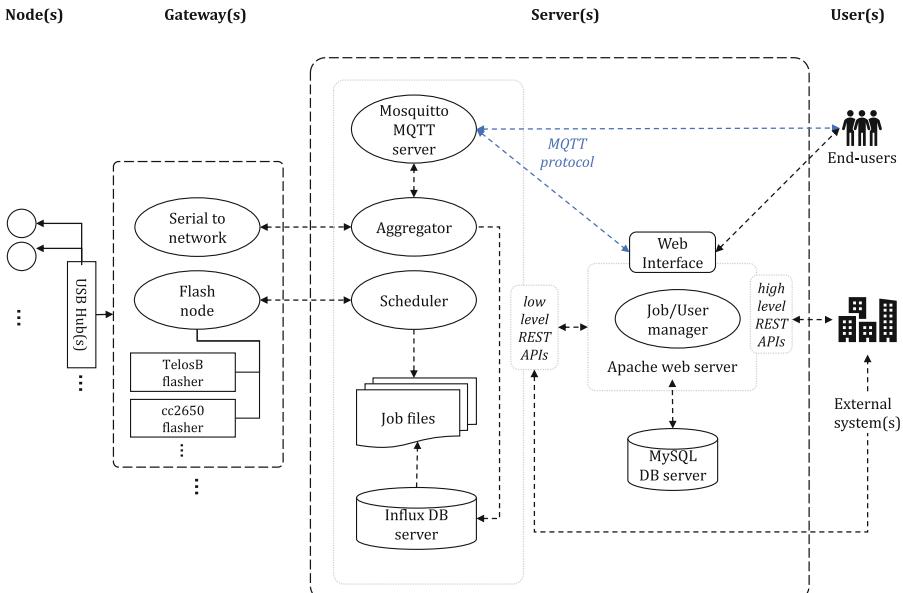


Fig. 1. System architecture of Indriya2

as deployed in the building. Figure 3 shows the gateway connected to the USB hubs and the active cables at the ceiling of the building.

4.2 Gateway Sub-components

The nodes are connected to the gateways via USB hubs. The gateways see the devices with unique names under linux directory system of “/dev/serial/by-id/”. For each type of device, the appropriate flasher application program is used to flash a particular device, based on its physical ID, with the user program. The flasher [5] used for flashing the TelosB can also be used for the Zolertia Z1 [6] and Arago Systems WisMote [7]. As for the SensorTags, the standalone flasher generated from the standard Uniflash interface [8], provided by the manufacturer (Texas Instrument), is used. It can flash other types of nodes from the same family including CC13xx, CC25xx, CC26xx, CC3220, CC3120, and others. Adding a new device category requires only adding, if needed, a new flasher program.

The data generated by the user programs are transmitted through the serial port to the network port using the **ser2net** program. The current version used [9] allows for multiple connections, whenever required, unlike the default version provided in the Linux installation. All the devices are uniquely mapped to a network port based on their physical ID. Table 1 shows the permanent ID used for the device as opposed to connection/temporary ID, which varies, as it is assigned at connection time. The table also shows excerpts for two types of devices, namely TelosB and CC2650, and their respective settings and configurations.

Given that a network port is associated with the serial data port of the device, a remote server program, namely the **aggregator** in our architecture, can connect to any selected node, identified by its network port and gateway IP address.

4.3 Server Sub-components

The main sub-components of the server are as follows:

Aggregator. The aggregator retrieves data from the nodes generated by the active job(s). One copy of the data is forwarded to the database (Influx DB) another copy is published to the appropriate active users via the MQTT server. The latter allows end-users to view in real-time, the output of the nodes. End-users can also push data to the nodes via the MQTT server. The facility to pull and push data enable applications that have both monitor and control requirements, include those that perform feedback control.

Scheduler. The scheduler starts/ends the jobs at the scheduled times. If the job is successfully started, the nodes’ status change to active and the aggregator starts forwarding the data to the data store. In case a node encounters a flashing failure, retries are performed up to three times. At the end of a job, the nodes’



Fig. 2. Motes as deployed in the building.

status is changed to inactive and the data stored for that job into the Influx DB is zipped and archived on the file system.

The Python Event Scheduler [10] was used, running on time resolution of microseconds.



Fig. 3. Gateway connected to USB hubs and active cables.

Table 1. Mapping device serial to TCP network port

Device type	Connection ID by device serial port in <code>/dev/tty*</code>	Permanent ID in <code>/dev/serial/by-id</code>	Map to TCP port	ser2net mapping configuration line
TelosB	<code>/dev/ttyUSB0</code>	<code>/dev/serial/by-id/usb-XBOW_Crossbow.Telos.Rev.B.XBTNPM52-if00-port0</code>	4000	<code>4000:raw:0:/dev/serial/by-id/usb-XBOW_Crossbow.Telos.Rev.B.XBTNPM52-if00-port0:115200,8DATABITS,NONE,1STOPBIT</code>
CC2650	<code>/dev/ttyACM0</code>	<code>/dev/serial/by-id/usb-Texas_Instruments.XDS110__02.03.00.08__Embed_with\CMSIS-DAP_L3002833-if00</code>	4100	<code>4100:raw:0:/dev/serial/by-id/usb-Texas_Instruments.XDS110__02.03.00.08__Embed_with\CMSIS-DAP_L3002833-if00:115200,8DATABITS,NONE,1STOPBIT</code>
	<code>/dev/ttyACM1</code>	<code>/dev/serial/by-id/usb-Texas_Instruments.XDS110__02.03.00.08__Embed_with\CMSIS-DAP_L3002833-if03</code>		

MQTT Server. The Mosquitto MQTT server [11] is used. The MQTT server allows the user or a remote program to process the data generated from the

user’s job in real-time. The user can also publish, i.e. send data, to a particular node which is part of his running job. While the MQTT ensures that legitimate users are allowed to publish, the aggregator makes sure that users only publish to node of their currently active jobs. The different levels of QoS supported by the server are **0** for at most once, **1** for at least once, and **2** for exactly once.

High Influx Database. Influx DB [12] is selected because of its ability to handle timestamped data at very high rate. Our evaluation shows that Influx DB meets our current and future requirements based on nodes capabilities to generate data at high rate. Note that sending records of timestamped data in batches, using JSON format over REST, allows for even higher data rate compare to writing records individually.

Job/User Manager. All users and respective job information are maintained using the MySQL DB. Upon user activation by the administrator, the former is given a limited quota of 30 min for scheduling jobs. Statistics on jobs scheduled can be viewed as well as editing jobs information and statistics is allowed.

REST APIs. The higher and lower levels of Indriya² are loosely-coupled as shown in Fig. 1. The low level REST APIs allows for any form of interface as long as REST calls can be performed. Moreover, even the front-end can be accessed with REST APIs. This allows for smooth federation with other testbed platforms in the future if needed. To enhance security while using the REST APIs, using the API requires adding a timestamp token to the job schedule request to protect against replay attack. Hash value of the request is calculated and appended which is used to preserve the integrity. The request is encrypted using the user’s MQTT credentials which preserve authenticity and confidentiality.

5 Executing a Job

In this section, we will describe the steps to create, schedule, and get the results from the job from the graphical user interface¹ as well as on the backend. After the user signs in, the user can create a job as shown in Fig. 4 where the user specifies the job name, chooses the mote-type, and selects the binary files to be used. In the next step, in Fig. 5, the user associates different binary files to different motes and creates the job. After creating the job, the user can schedule the job in a free time slot as shown in Fig. 6.

When it is time to execute a particular job, the scheduler locks the job so that its cancellation cannot be executed on that particular job during that time it is going to be flashed. This is done because once the flashing procedure is launched on the device, it cannot be interrupted. For each node to be flashed, a thread is launched that firstly uses **rsync**, without overwriting, to copy the

¹ <https://indriya.comp.nus.edu.sg/>.

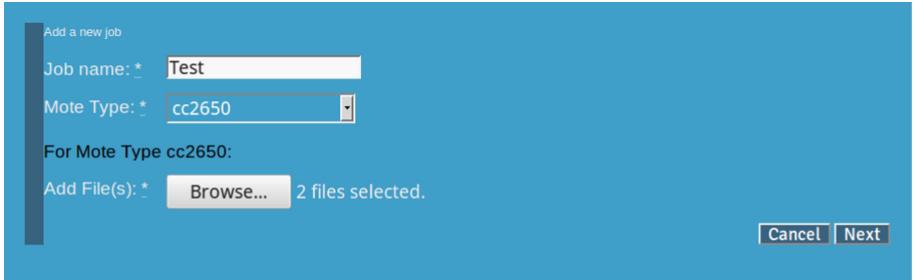


Fig. 4. Start creating a job.

required binary file to the respective gateway. Secondly, `ssh` is used to launch the appropriate flasher command with the required parameters to burn the binary files to the targeted nodes. The process is depicted in Fig. 7.

When an active user sends data to a node of his running job via the MQTT server, the latter checks whether the user is legitimate and the aggregator checks whether the running node is under that particular active user. The data is then pushed to the node by the aggregator through the network connection.

Finally, at the end of the job, the scheduler executes functions to: (1) set the status of the active users/nodes lists, so that the aggregator stops forwarding activities for the users/nodes related to that job, (2) retrieve the job's data from the Influx DB, zip and save it on the file system so that it can retrieved by the web server, and (3) run a maintenance job that flashes a default program to reset the node to a default behavior.

6 Evaluation

The wiring infrastructure of Indriya2 allows the installation of more than 140 sensor nodes. In this evaluation, the configuration has 102 motes out of which 74 are TelosBs and 28 are CC2650 sensortags. TelosB comes with an 8 MHz TI MSP430 microcontroller with 10 KB RAM and most of them have light, temperature and humidity sensors integrated. The CC2650 sensortag has an ARM Cortex-M3, reaching up to 48 MHz, with 28 KB SRAM. It has more sensors, including barometer, microphone, gyroscope, accelerometer, magnetometer by default.

In this section, we present two sets of experimental results. In the first set of experiments (Sect. 6.1 to 6.3) we present results on the network connectivity across the entire network as well as within each device cluster. In Sect. 6.4, we look at the performance of the MQTT server.

In order to measure network connectivity, we program each node to broadcast one packet every second for a total of 100 s each in a round robin fashion. When a node is not transmitting, it listens to the channel and records the packets, including the source of the packet and the received signal strength (RSS) of each packet. All packets are transmitted on channel 26. As each packet contains

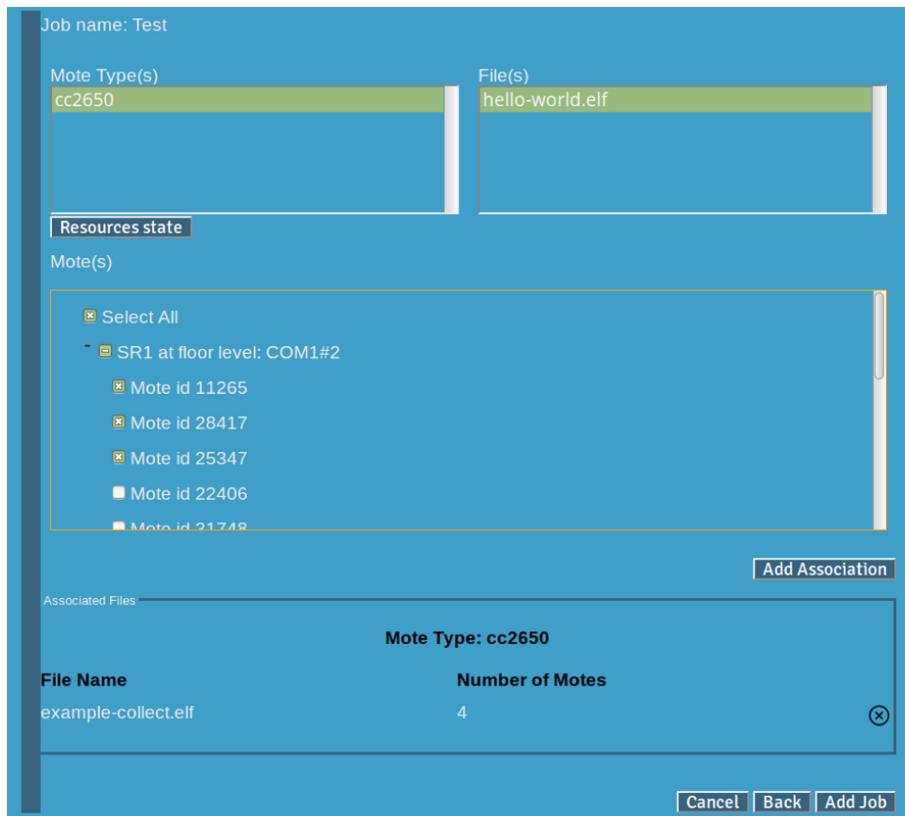


Fig. 5. Associate binary files to motes.

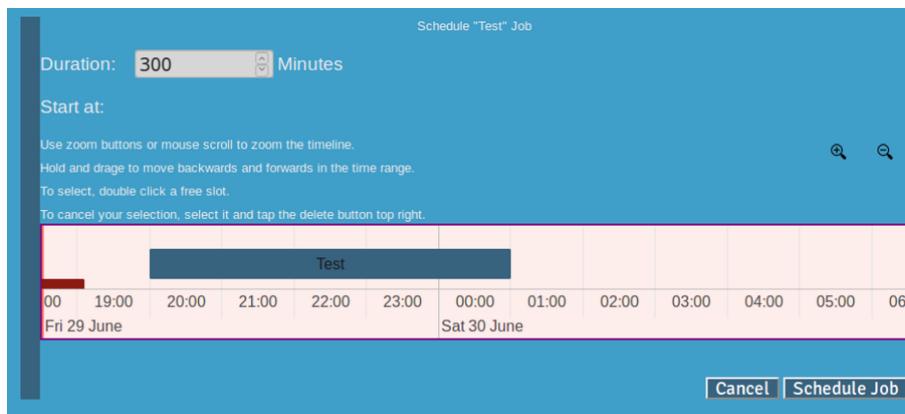


Fig. 6. Schedule a job.

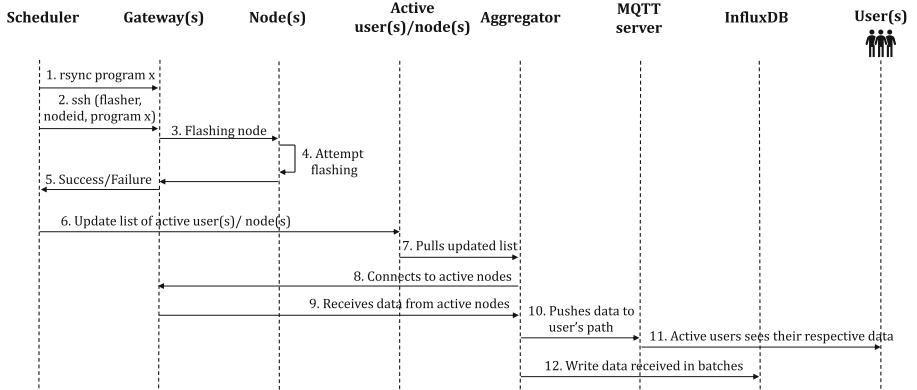


Fig. 7. Job execution process

a sequence number and we know the total number of packets transmitted by each node, we can compute the packet reception ratio (PRR) of the wireless link between two nodes.

6.1 Nodes' Connectivity

The network connectivity measured is shown in Fig. 8. Blue squares and red circles are used to denote CC2650s and TelosBs respectively. We used black solid links to indicate links with PRR greater than or equal to 0.8 and dotted lines for links with PRR between 0.5 and 0.8. Links with PRR less than 0.5 are not shown.

Tables 2 and 3 summarize the connectivity in different forms, considering links with different PRRs. We can make the following observations. First, connectivity is clearly not symmetric as in-degrees are different from out-degrees. Considering the case for $PRR > 0.8$, the median connectivity for TelosB and CC2650 are 6 and 4 respectively.

6.2 PRR and RSS Distribution

The overall link quality of the different clusters, based on PRR and RSS, are shown in Fig. 9. The skewness of the PRR distribution for both the CC2650 and TelosB clusters indicate that the link quality is either high or low. The RSS observed by the TelosB cluster is clearly centered at a lower dBm, while in general, the RSS observed is highest around -85 dBm.

6.3 Correlating Performance (PRR) with RSS

The correlation between RSS and PRR is useful for protocol that uses RSS to estimate the link quality (PRR). From Fig. 10, both the TelosB and the CC2650 clusters shows a clear correlation between higher PRR with higher RSS. However, it is also noted that there are some cases where the RSS is low but PRR is still high.

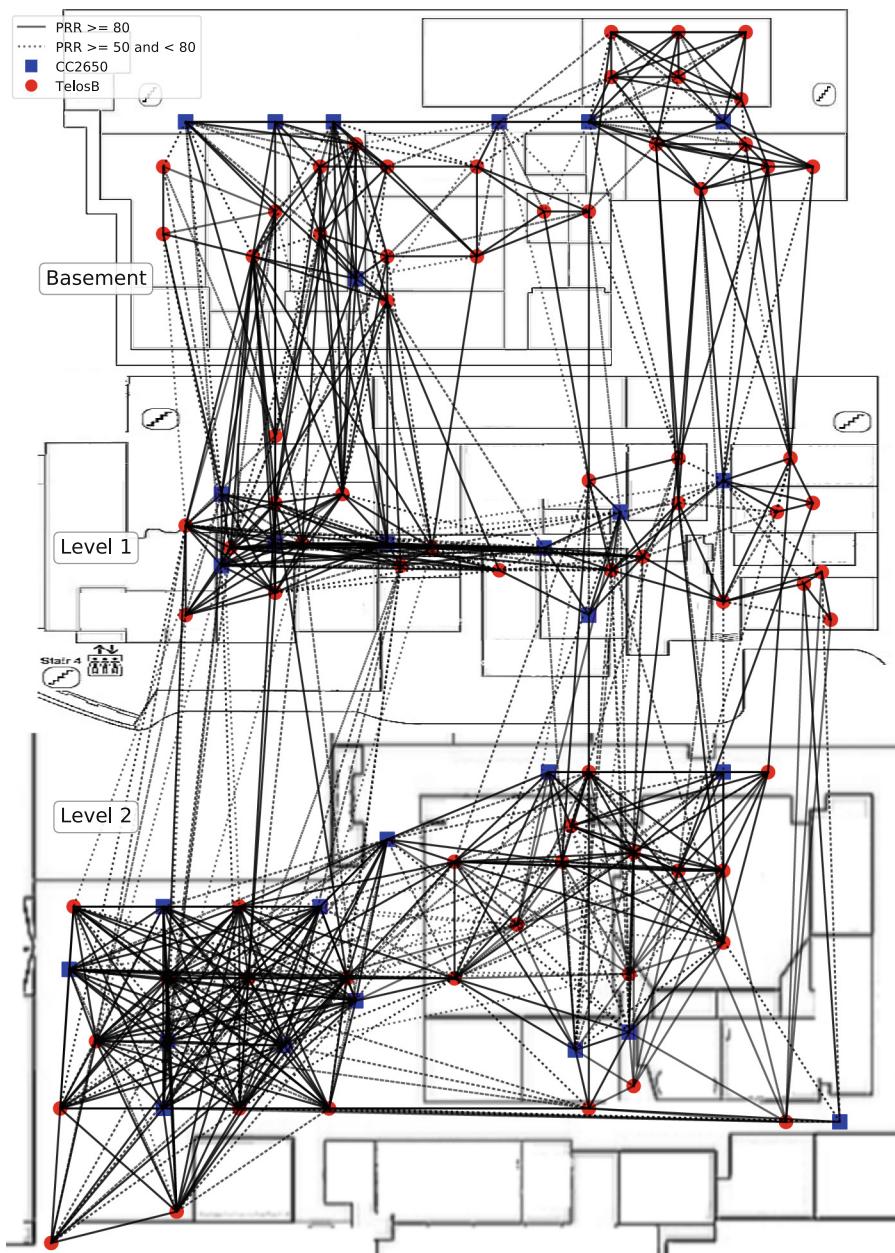


Fig. 8. Indriya \mathcal{Z} : Layout of the heterogeneous testbed.

Table 2. Nodes connectivity (PRR > 80)

Cluster		TelosB (74)	CC2650 (26)	Heterogeneous (102)
Link type	Percentile			
In-degree	10th percentile	3	1.5	4.1
	Median	6	4	9
	90th percentile	10	6	17
Out-degree	10th percentile	1.3	2	3.1
	Median	6	4	10
	90th percentile	11	7	17
Cluster density (0-1)		0.087	0.157	0.096

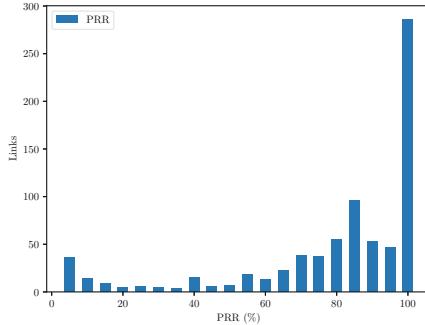
Table 3. Nodes connectivity (PRR > 50)

Cluster		TelosB (74)	CC2650 (28)	Heterogeneous (102)
Link type	Percentile			
In-degree	10th percentile	5	2	7
	Median	9	5	13
	90th percentile	13	8	20.9
Out-degree	10th percentile	3.3	2	5
	Median	9	5	13
	90th percentile	14	7	21.9
Cluster density (0-1)		0.123	0.187	0.136

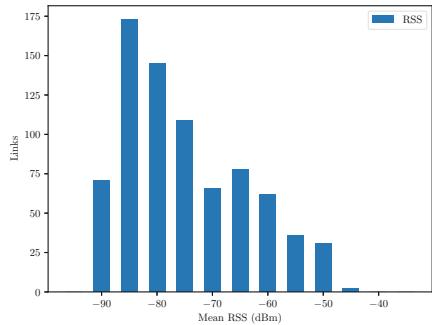
6.4 Benchmarking for Scalability

Firstly, to evaluate the performance of the MQTT server, two tests were performed. In the first test, we want to find out how scalable is the testbed infrastructure as more nodes are added and each node may transmit at high data rate. Currently, a node on the testbed can generate on the average around 11,500 bytes (115 lines/samples of 100 bytes) on the serial port. The benchmarking tool from [13] was used. Evaluation shows that the system can support more than 500 nodes, each sending 115 messages of 100 bytes to our MQTT server.

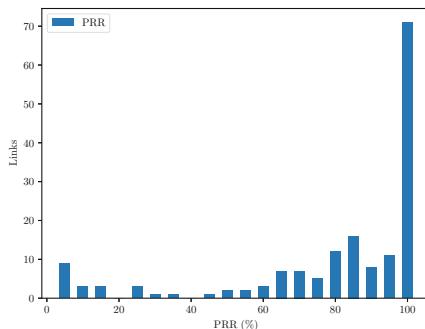
In the second test, we compare the performance of MQTT subscription against a direct TCP connection with nodes generating data samples at approximately 120 Hz. This test compares the performance of using the publish/subscribe feature provided by MQTT to the approach of fetching data directly using TCP sockets. While the TCP connection carries only the data generated by the sensor (for the test we used $<\text{loadtest}, \text{nodeid}, 7041, 1, \text{this is a load test}>$), the content from our MQTT broker also adds some metadata, namely the timestamp and the node id of the source, as seen in the following



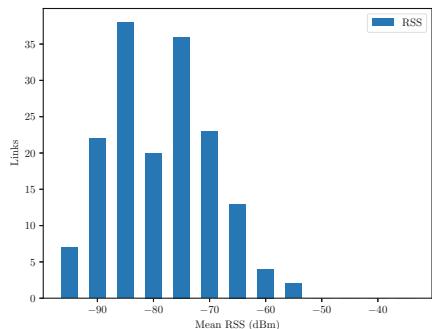
(a) PRR distribution - TelosB Cluster



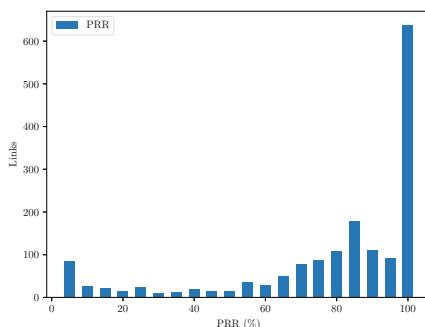
(b) RSS distribution - TelosB Cluster



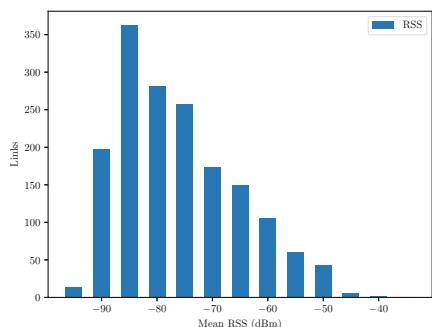
(c) PRR distribution - CC2650 Cluster



(d) RSS distribution - CC2650 Cluster



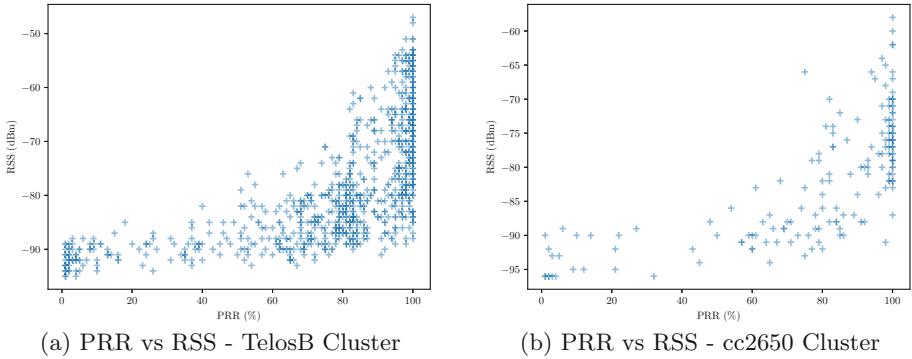
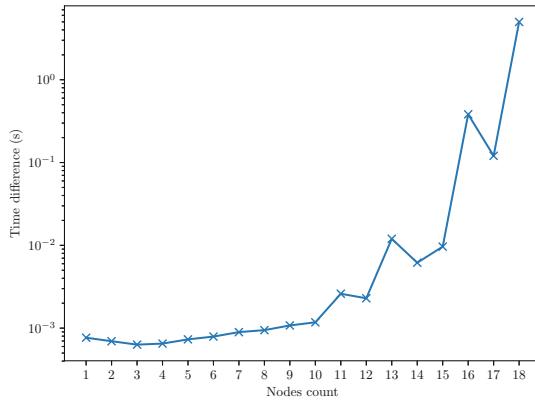
(e) PRR distribution - Heterogeneous Cluster



(f) RSS distribution - Heterogeneous Cluster

Fig. 9. PRR and RSS distribution for the TelosB cluster, CC2650 cluster and both combined as a heterogeneous cluster using the IEEE 802.15.4 standard.

<“nodeid”, “7041” “value”: “loadtest,nodeid,7041,1, this is a load test”, “time”: “2018-06-29T12:35:45.731053Z”>. In this experiment, the relative delay using the two modes of connection, to receive/subscribe to the data, are shown in Fig. 11.

**Fig. 10.** PRR vs RSS**Fig. 11.** Average delay between messages from MQTT subscription messages and direct TCP socket

In the evaluation, every two minutes, a node is added to the group of nodes that are generating data to the MQTT server and the average time difference taken over each sample received over the period preceding the arrival of the next node is computed and plotted. It is noted that after the arrival of the 16th and 17th nodes, the average time difference is still less than 500 ms. At that time, the total number of messages generated and received in one second is over 2000. This implies that the testbed can easily support 200 nodes generating data at 10 Hz, which is sufficient for most wireless sensor network applications.

For the evaluation of the Influx database, nearly 300 millions messages were generated and successfully recorded over a period of 4 h. This means that the system can support 200 nodes generating data at 100 Hz over a period of 4 h.

7 Conclusion

The migration of Indriya to Indriya2 allows for more features. The testbed supports heterogeneous nodes and usage of different wireless network technologies. The new architecture, with REST APIs, allows the system to be federated with other testbeds fairly easily. In the IoT era, Indriya2 is equipped with the lightweight protocol, MQTT, and a database allowing for high influx of time series data. Benchmarking results show that Indriya2 can easily scale up to a large number of nodes.

References

1. Doddavenkatappa, M., Chan, M.C., Ananda, A.L.: Indriya: a low-cost, 3D wireless sensor network testbed. In: Korakis, T., Li, H., Tran-Gia, P., Park, H.-S. (eds.) TridentCom 2011. LNICST, vol. 90, pp. 302–316. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29273-6_23
2. TinyOS-main. <https://github.com/tinyos/tinyos-main/releases>. Accessed 4 June 2018
3. Multi-Standard CC2650 SensorTag Design Guide. <http://www.ti.com/lit/ug/tidu862/tidu862.pdf>. Accessed 4 June 2018
4. TelosB Mote Platform. https://www.willow.co.uk/TelosB_Datasheet.pdf. Accessed 4 June 2018
5. Python-msp430-tools for TelosB, Z1 and WisMote. <https://github.com/cetic/python-msp430-tools>. Accessed 5 June 2018
6. Z1 Datasheet. http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf. Accessed 5 June 2018
7. WiSNet: Complete solution for IPv6 sensor networks. <http://www.aragosystems.com/produits/wisnet/wismote/>. Accessed 5 June 2018
8. Uniflash Standalone Flash Tool for TI Microcontrollers (MCU), Sitara Processors & SimpleLink devices. <http://www.ti.com/tool/UNIFLASH>. Accessed 5 June 2018
9. Serial port to network proxy enhanced for multi-connections support! <https://github.com/nickxia/ser2nets>. Accessed 4 June 2018
10. Sched - Event scheduler. <https://docs.python.org/3/library/sched.html>. Accessed 4 June 2018
11. Eclipse MosquittoTM: An open source MQTT broker homepage. <https://mosquitto.org/>. Accessed 4 June 2018
12. InfluxDB homepage. <https://docs.influxdata.com/influxdb>. Accessed 4 June 2018
13. Simple MQTT (broker) benchmarking tool. <https://github.com/krylovsk/mqtt-benchmark>. Accessed 30 June 2018
14. Lim, R., Ferrari, F., Zimmerling, M., Walser, C., Sommer, P., Beutel, J.: FlockLab: a testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In: Proceedings of the 12th International Conference on Information Processing in Sensor Networks, New York, pp. 153–166 (2013)
15. Boano, C.A., Voigt, T., Noda, C., Römer, K., Zúñiga, M.: JamLab: augmenting sensornet testbeds with realistic and controlled interference generation. In: Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks, pp. 175–186 (2011)
16. Adjih, C., et al.: FIT IoT-LAB: a large scale open experimental IoT testbed. In: IEEE 2nd World Forum on Internet of Things (WF IoT), pp. 459–464 (2015)

17. Handziski, V., Köpke, A., Willig, A., Wolisz, A.: TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In: Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: From Theory to Reality, pp. 63–70. ACM (2006)
18. Werner-Allen, G., Swieskowski, P., Welsh, M.: MoteLab: a wireless sensor network testbed. In: The Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, p. 68 (2005)
19. Levis, P., et al.: TinyOS: an operating system for sensor networks. In: Weber, W., Rabaey, J.M., Aarts, E. (eds.) Ambient Intelligence. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-27139-2_7
20. Dunkels, A., Gronvall, B., Voigt, T.: Contiki-a lightweight and flexible operating system for tiny networked sensors. In: Local Computer Networks (2004)