Towards Trusted Seamless Reconfiguration of IoT Nodes

Full Paper

Tony Toms
School of Information Technology
Deakin University
Geelong, Australia
Email: tonytoms01@gmail.com

Kevin Lee
School of Information Technology
Deakin University
Geelong, Australia
Email: kevin.lee@deakin.edu.au

Abstract

IoT networks are growing rapidly with the addition of new sensors, nodes and devices to existing IoT networks. Due to the ever-increasing demand for IoT nodes to adapt to changing environment conditions and application requirements, the need for reconfiguring these already existing IoT nodes is increasing rapidly. A reconfiguration of an IoT network includes alterations to the devices connected, changing the behavioural patterns of the devices and modifying the software modules that control the IoT network and devices. Reconfiguring an already existing IoT network is a challenge due to the amount of data loss and network downtime faced when carrying out a reconfiguration procedure in a limited power supply environment. This paper proposes an architecture for trusted dynamic reconfiguration of IoT nodes with the least amount of data loss and downtime. The proposed approach uses multiple IoT nodes to facilitate dynamic reconfiguration.

Keywords - IOT, reconfiguration, middleware, seamless data collection

1 Introduction

The Internet of Things (IoT) is becoming a major trend in computing and is having a high impact by solving real-world problems such as improving the energy efficiency of IT rooms, enabling smart cities and supporting logistics. The core idea behind IoT is embedding tiny, networked, electronic devices into the surrounding physical objects (Iwanicki 2018), thus enabling the everyday real-world objects to interact with each other. As per recent studies, by the year 2020, there would be around 50 billion sensors and devices connected to the internet (Zhou 2016). As IoT networks become larger and larger, an increasing concern is the degree of heterogeneity and need to evolve IoT deployments.

In an IoT network, nodes and sensors are commonly not from a single vendor. A typical IoT network is equipped with multiple types of devices and sensors that are communicating directly or via the Internet. This increased degree of heterogeneity has encouraged the development of flexible IoT networks. A flexible IoT network would discover these heterogeneous devices dynamically, update the software required for the devices and provide an updated end user application (Heo 2015).

Flexible IoT enables the addition of a nodes and sensors after deployment whilst the network is operating. They provide agents that enable the networks to handle various communication protocols and data formats used by different devices (Pazos 2015). However, changes happening to an IoT networks won't be limited to the communication requirements for just a family of devices. An IoT network would eventually face situations where a change in the behavioral patterns of the devices are to be done or a newly introduced device must be incorporated. Further changes to the network would include updating the core software modules in different nodes, changing node responsibilities or updating the node behavior.

A flexible IoT network would be capable of handling the any deployment reconfiguration. To do this, requires that IoT nodes and IoT networks are dynamic reconfigurable. A reconfigurable IoT network would enable the network to handle all these changing network requirements. It would enable the networks to accept new requirements by allowing computation and communication services to evolve over time with minimal disruption (Raagaard 2017). The main advantage of making an IoT reconfigurable is that it allows functionality to be altered without much network downtime or data loss (Pena 2017).

For IoT networks to adapt to changing conditions, a reconfigurable IoT node is needed. The main advantage of making an IoT node reconfigurable is that it allows functionality to be altered without any physical change for the systems (Pena 2017). Thus, making a reconfigurable IoT node would enable the entire IoT systems to be dynamic and easy to maintain.

This paper proposes a new method to enable trusted dynamic reconfiguration of IoT networks without compromising the network downtime and providing seamless data collection from the nodes throughout the reconfiguration. Reconfiguration is carried out through a middleware that updates nodes and then dynamically switches the running application programs from the old versions to the newly updated versions with least data loss. To evaluate this approach, a prototype was built using Raspberry Pi's to act as the IoT nodes and a middleware module is written in Python.

The remainder of the paper is as follows. Section II provides a background on reconfiguration in the Internet of Things (IoT). Section III proposes a new architecture for seamless IoT reconfiguration. Section IV describes the implementation of the architecture. Section V presents an evaluation of the proposed architecture. Section VI presents some conclusions and discusses future work.

2 Background

2.1 IoT Network Overview

An IoT network can be represented as layered architecture, as illustrated in Figure 1. The top most layer, which is the application layer consist of application software and middleware software. The nodes and devices are connected to the application layer via the physical layer objects like WIFI, Bluetooth, LTE, Internet etc. The system layer consists of system kernel and UDP packets belong to the transport layer. The network protocols and the gateways belong the network layer.

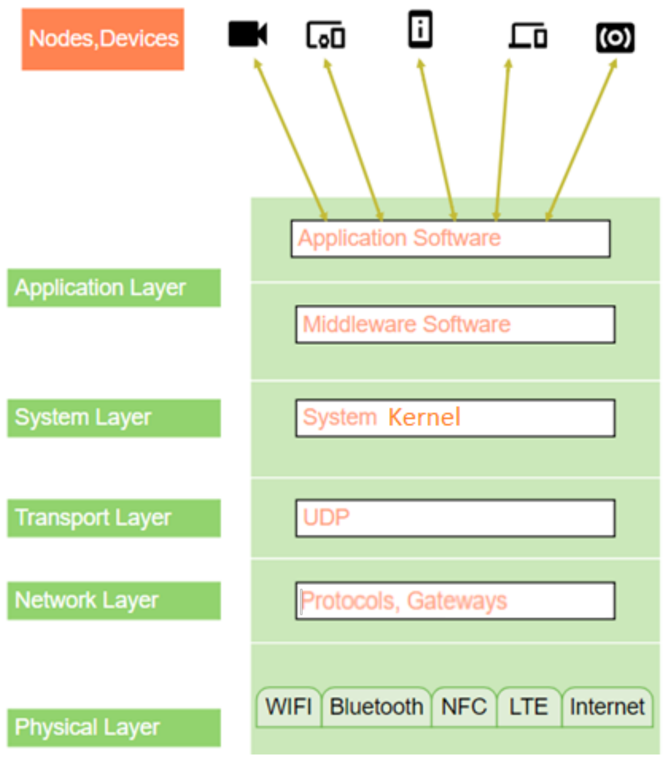


Figure 1. IoT Network Layered Structure

2.2 IoT Structural and Behavioural Reconfiguration

Reconfiguring a network can be broadly classified as either i) structural reconfiguration or ii) behavioural reconfiguration. The structural reconfiguration deals with replacing a section or a function of the IoT network where as a behavioural reconfiguration deals with changing the way sensors behave in different scenarios by changing the properties or parameters.

One of the most common approach in behavioural reconfiguration is the use a monolithic image which is a collection of all the drivers, middleware and the core software modules. Any changes performed to the network is by the use of monolithic image where the middleware would completely change all the data and replace it with the new image. This approach is flexible but consumes a lot of energy due to the large data transfer. Due to the large quantity of data consumption, a better approach to the behavioural reconfiguration is adopted where the system loads only the updated software modules and link them to the devices during the runtime.

In a behavioural reconfiguration, the modules modify the properties and change the connections between the components during the network runtime. This approach provides an efficient mechanism for reconfiguration as it causes the least amount of resource utilization and data transfer. However, this type of reconfiguration is extremely narrow in terms of the functionality it provides.

The key aspect in providing an effective reconfiguration strategy is to adopt the positive sides from both runtime reconfiguration and behavioural reconfiguration.

2.3 Recent approaches for Dynamic reconfiguration

An approach to dynamically add sensors to IoT network is proposed by (Boman 2014) by using GSN, Firebase and IoT data interpreter. By using GSN and virtual sensors the authors where able to add any number of sensors as they want into their network. The data interpreter uses an XML file as an input to see the status of all the devices connected. Firebase, which is a real-time NoSQL database management system owned by google is used as the data storage to enable interfacing with 3rd party application (Prado 2017). Even though the system handles the addition and removal of sensors, they won't deal with the downtime faced when the core application modules needs an update.

(Myint 2017) proposes a wireless sensor network-based system for monitoring water quality. In the system architecture, sensors like Ph sensor and ultrasonic sensors are connected to a FPGA system

(Hauck 1994). The software applications for the sensors are written in C and is run using an Eclipse IDE. Here the system is said to be reconfigurable using a “. sopcinfo” file, which saves all the configuration information. Here the system can only alter parameters like time interval of monitoring, temperature reading formats etc. That is, this architecture deals with the behavioural change (Sarray 2015) and doesn’t deal with software/hardware reconfiguration.

(Lu 2016) proposes an IoT inventor which is a web-based composer for reconfigurable agentized services. Here the authors talk about temporarily reconfiguring the services based on user interest. That is instead of tightly coupling sensors, they use sensors to infer context of interest. Here the paper deals with changing the context in which the sensors work and is not dealing with modifying the IoT network as such.

More work in this area is proposed by (Chi 2014). Here the authors use CPLD designed based on IEEE1451. Since the components belong to IEEE1451 family they are interoperable (Becari 2016). There the proposed system can identify newly connected devices intelligently. Here all the drivers of the sensors which are to be used must be written into the CPLD. In case of a new sensor, the system must go down and an update is required.

There have been substantial efforts in virtualization and cloud computing in the reconfiguration of virtual machines and cloud instances (Anton 2010, Ali 2010). There is also considerable effort in adaptively scheduling jobs on Cloud resources (Lee 2011, Lee 2009). This is supported by dynamically reconfigurable networking infrastructure (lee 2006). These approaches are in situations with fully controlled environments and non-resource constrained. This is opposed to IoT deployments, which have difficult deployment scenarios and are resource constrained. Thus, there is still a need for domain-specific reconfiguration solutions for the Internet of Things.

3 A Trusted Architecture for seamless IoT reconfiguration

3.1 Design Overview

The main idea in this proposal is to use multiple nodes to facilitate IoT dynamic reconfiguration with seamless data collection from the sensors, as illustrated in Figure 2. A controller node is used to control the activities of these nodes.

The controller manages the activities of the connected nodes. At any point in time one of the nodes is active and collecting the data from all sensors. During a reconfiguration request from the main controller, the control is changed to the backup node. The standby node starts up and collect data from the sensor and is send to the cloud storage. The main node would stop execution for undergoing the updates. Once the software deployment activities and done, the main node programs are fired up and the backup node goes to idle state.

All the executables which are fired up at the node side would be running on top of the proposed middleware. The middleware also handles all the main controller activities. Different communication protocols are used by the middleware to send files between the controller and the nodes.

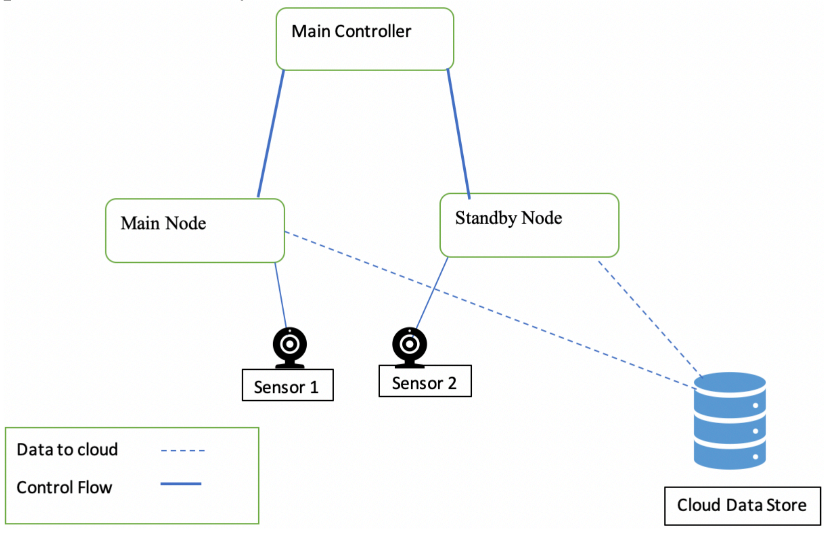


Figure 2. The Node Arrangement for the Architecture

3.2 Architecture

Figure 3 illustrates a middleware software architecture for enabling the seamless reconfiguration of IOT nodes. All the programs which are to be executed, the data to be transferred and all the communications between the nodes are handled by the middleware. All the programs and software run on top of this middleware. The middleware also provides a platform for the dynamic deployment of software modules into these nodes. A Cloud-based MQTT, which is a light weight message queuing protocol (Kodali 2016), is used as the Cloud broker to obtain data.

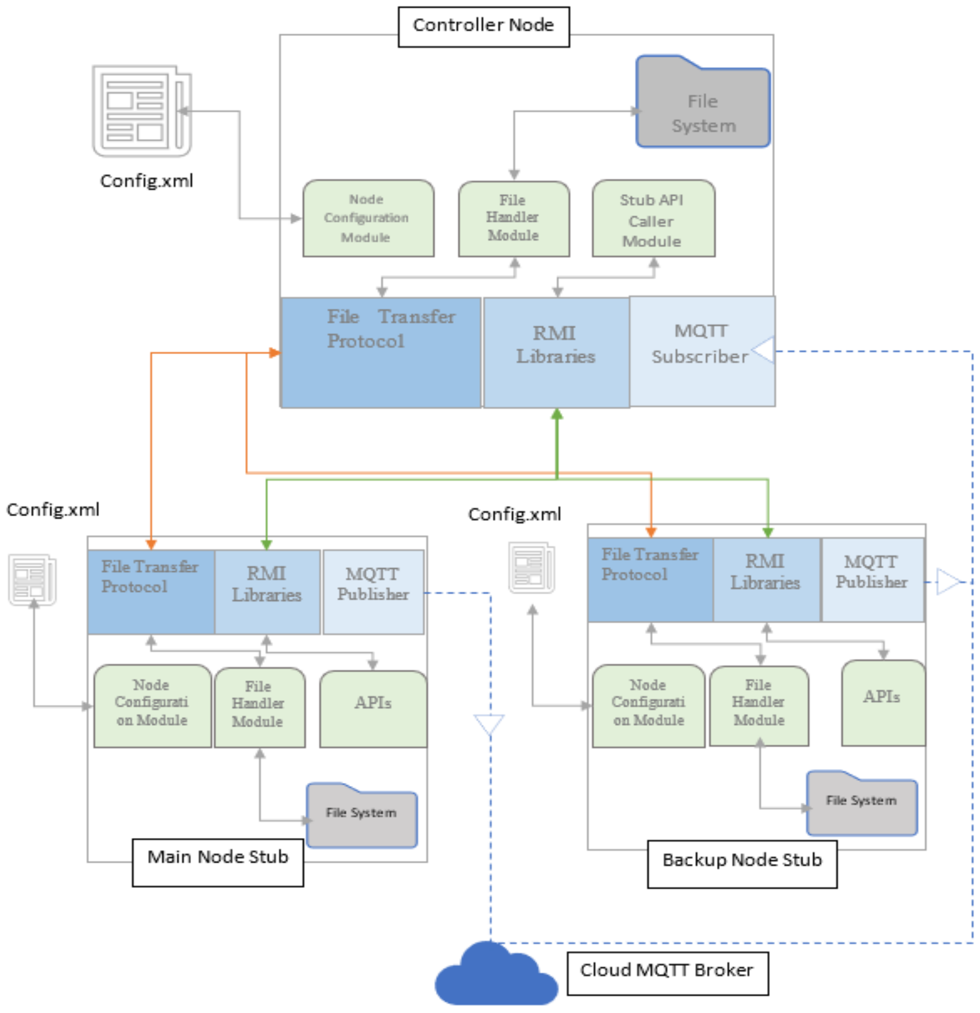


Figure 3. The Middleware Architecture

This architecture can be broadly classified into 3 versions namely, the controller version, the main node version and the backup node version. The controller node version is implemented at the main controller node, the node version at the main node and the backup node version at the back up node respectively. The modules inside each of these versions can be categorized into three levels, as detailed in Figure 4.

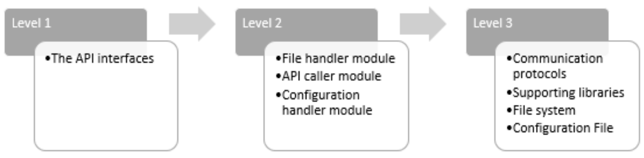


Figure 4. Module Levels in each Version of the Middleware

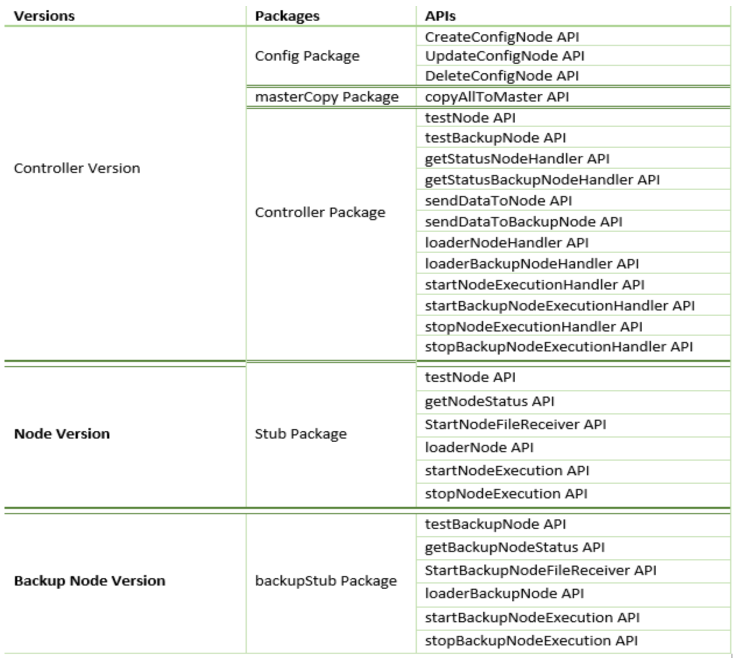
Users directly interact with the API interfaces. All the reconfiguration needs like addition of nodes, deletion of nodes, file transfer, node-side method invocation callers and status check can be achieved using the API interfaces. All these facilities are exposed to the world as API interfaces.

The second level contains all the modules inside the middleware. The file handler module handles all the API requests related to file management. The API caller module is used to remotely call the APIs of node and backup node from the controller node. The configuration handler module handles all the configuration needs of the middleware.

The third level contains all the libraries and protocol handlers. Different protocols are used for remote method invocations, remote file distribution and control transfer.

3.3 List of APIs

At the controller version, there are mainly 3 packages namely the config package, the controller package and the masterCopy package. At the node version there is only 1 main package called the stub package. At the backup node version, the main package name is backupstub package.



Versions	Packages	APIs
Controller Version	Config Package	CreateConfigNode API
		UpdateConfigNode API
		DeleteConfigNode API
	masterCopy Package	copyAllToMaster API
	Controller Package	testNode API
		testBackupNode API
		getStatusNodeHandler API
		getStatusBackupNodeHandler API
		sendDataToNode API
		sendDataToBackupNode API
		loaderNodeHandler API
		loaderBackupNodeHandler API
		startNodeExecutionHandler API
		startBackupNodeExecutionHandler API
		stopNodeExecutionHandler API
		stopBackupNodeExecutionHandler API
Node Version	Stub Package	testNode API
		getNodeStatus API
		StartNodeFileReceiver API
		loaderNode API
		startNodeExecution API
		stopNodeExecution API
Backup Node Version	backupStub Package	testBackupNode API
		getBackupNodeStatus API
		StartBackupNodeFileReceiver API
		loaderBackupNode API
		startBackupNodeExecution API
		stopBackupNodeExecution API

Table 1. List of all the APIs in the middleware

In total there are 28 APIs, a listed in Table 1, out of which 10 are API handler. An API handler is an API that calls other APIs remotely.

The APIs in config package is used to alter the details on configuration xml file. The 'CreateConfigNode' API is used to add new nodes details into the configuration xml file. The 'UpdateConfigNode' API is used to update the details of the nodes and the 'DeleteConfigNode' is used to delete the node details from the configuration file

The 'masterCopy' package contains a single API called the 'copyAllToMaster' API. This API is used to copy the updated software to the controller for copying it to corresponding nodes.

The controller package contains 2 kinds of APIs. The normal APIs and handler APIs. The 'testNode' API and 'testBackupNode' API would check if the node and backup nodes are up and running. 'sendDataToNode' would send the data (the updated software) to the node and the 'sendDataToBackupNode' would send the updated software to backup node.

The rest of the APIs in controller package are handler APIs which are used to call other APIs from the node and backup node. The 'getStatusNodeHandler' would call the 'getNodeStatus' from the main node. This API is used to get the current node status. Similarly, 'getStatusBackupNodeHandler' would call the 'getBackupNodeStatus' which would give the status of the backup node to the controller.

The 'loaderNodeHandler' API and 'loaderBackupNodeHandler' would invoke the 'loaderNode' from the node and 'loaderBackupNode' from the backup node respectively. These APIs would load the received updated software from the received folder to corresponding execution folder.

The 'startNodeExecutionHandler' would call the 'startNodeExecution' from the node. This API would execute files from the execution folder whose names are listed in the configuration file's executables list. Similarly, 'startBackupNodeExecutionHandler' would call the 'startBackupNodeExecution' from the backup up node, which would start executing the files at the backup node.

'stopNodeExecutionHandler' would call the 'stopNodeExecution' which in turn would stop the node execution. 'stopBackupNodeExecutionHandler' would call the 'stopBackupNodeExecution' which in turn would stop the backup node execution.

4 Implementation

4.1 Overview

To test the architecture, an implementation was built using Raspberry Pi nodes and Grove PI+ sensors. A Raspberry Pi acts as the controller and two other Raspberry Pi's act as the node and backup node. Sensors are connected to the Node and backup node Raspberry Pi. The idea is to check if a continuous and seamless data collection from these sensors can be achieved during a reconfiguration process.

To test this, the Raspberry Pi's are first implemented with sensors and other required software along with the proposed middleware. The setup would be working, and the data collected would be stored on to the cloud data storage. Then a reconfiguration request is initiated. During the process, the middleware would switch the control from the main node to the backup node and check if backup node starts fetching the data from the sensors by checking the cloud storage. Once the backup node started successfully, the execution is stopped at the main node. The updated software and programs are then sent to the main node. Once the files are received, the loaderNode API is used to load the files and then the execution of the updated software module would be started. Once the main node starts successfully with the updated software, the backup node is stopped. The data collected at the cloud data store can be analyzed to check if data reading is seamless.

4.2 Requirements

A temperature and humidity sensor is used in this implementation. The data from the temperature and humidity sensor is being collected from the sensor. The temperature is read in degree Celsius. The reconfiguration request is be change it from degree Celsius to Fahrenheit reading. The updated software module should be deployed on to the devices dynamically with no network downtime.

During this reconfiguration change, following requirements should be strictly followed to check if this architecture achieved its goals.

Requirement 1: Data should be continuously collected to the cloud storage throughout the reconfiguration without any delay (Seamless Data Collection).

Requirement 2: There should be no network downtime when carrying out the reconfiguration. (No Network Downtime).

Requirement 3: The files should be sent remotely without any physical contact with the nodes. (Remote Reconfiguration)

Requirement 4: The node after reconfiguration should successfully run the updated software module. (software Deployment)

4.3 Physical Setup

To test the working of the algorithm, 3 Raspberry Pi's are used. All the Raspberry PI nodes are connected to the same network using a router. One of the Raspberry Pi nodes act as the controller node and would control other Raspberry Pi nodes. The other two Raspberry Pi nodes are installed with node version and backup node versions. The picture in Figure 5 illustrates the experimental setup.

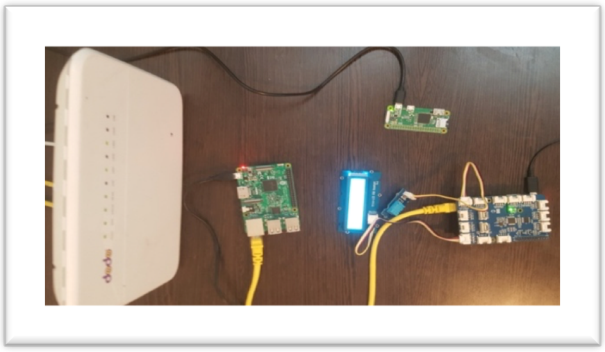


Figure 5. The Experiment setup

The diagram for the setup is given in Figure 6. The controller node is connected to the network via WIFI. The main node and the backup node are connected via RJ45 cables. The sensor is connected to the port D4 and the LCD display is connected to port 12C.

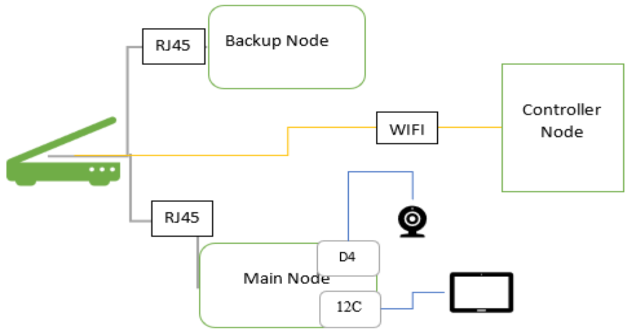


Figure 6. Overview of the IoT network.

4.4 Initial Setup

The application for reading the temperature and humidity is up and running on to the main node. The Application is executed on top of the middleware software module at the main node side. The application reads the values at an interval of 2 seconds each. The values are displayed on the LCD screen connected along with publishing the collected values into the MQTT broker (Singh 2015) each time.

The MQTT broker used here is having the host name as "test.mosquitto.org". The values are published to "seamlessIOTReconfig/data". The value is having the following format:

"System Time, IP Address, Temp= Value, Humidity = Value"

A client program on a PC connected to the network subscribes to the node and data is logged to log.txt. This is done to check the data is collection frequency and to test if there is any network downtime.

4.5 Reconfiguration Steps

The general flow of reconfiguration process is listed below.

step 1: Update the main configuration file. Called the API *UpdateConfigNode()*

step 2: Copy the updated program to controller node. Called the API *CopyAllToMaster()*

step 3: Start the backup node. Called the API *startBackupNode ExecutionHandler()*

step 4: Stop the main node. Called the API *stopNodeExecution()*

step 5: Send data to main node. Called the API *SendDataToNode()*

step 6: Load the updated program onto main node. Called the API *loaderNode()*

step 7: Start execution of main node. Called the API *startNodeExecution()*

step 8: Stop the backup node. Called the API *stopBackupNodeExecution ()*

step 9: Send data to the backup node. Called the API *sendDataToBackupNode()*

step 10: Load data to the backup node. Called the API *loaderBackupNode()*

4.6 Result

The reconfiguration was successful, The LCD starts showing the temperature in Fahrenheit value. Figure 7 shows the LCD output. The LCD on top shows the reading in Celsius, before the reconfiguration. The LCD on the bottom shows the reading in Fahrenheit, which is after the reconfiguration.

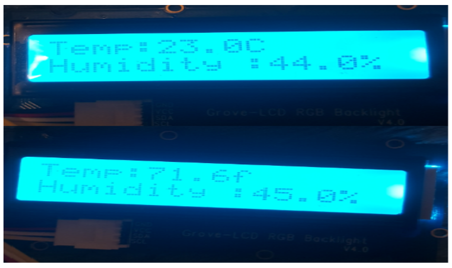
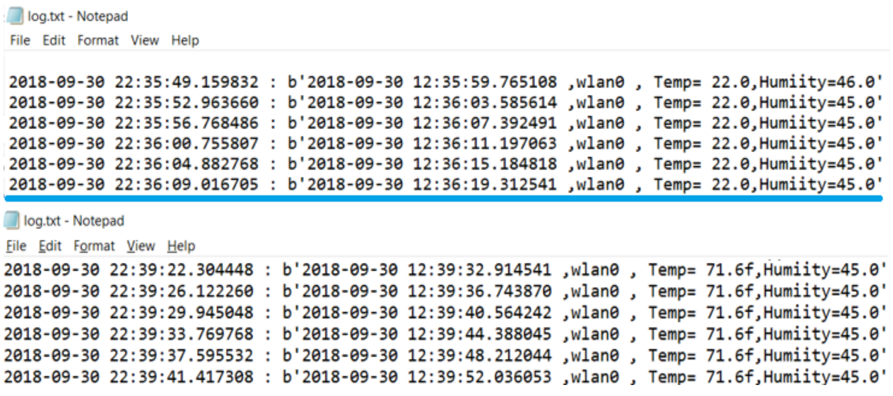


Figure 7. The LCD Output

5 Evaluation

To achieve the goal of reconfigurable IoT, all the 4 requirements namely, seamless data collection., no network downtime and remote reconfiguration, software deployment should be achieved. The proposed architecture is implemented as described in Section 4.3. Primary evaluation is performed on the data collected by the device, which is collected into a log.txt file on a client computer.

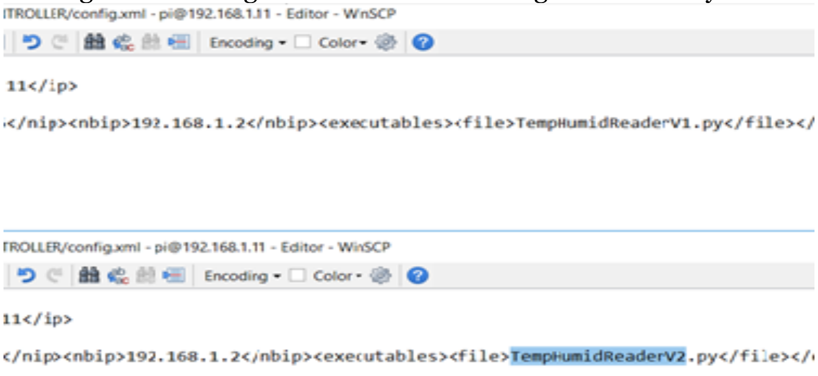


```
2018-09-30 22:35:49.159832 : b'2018-09-30 12:35:59.765108 ,wlan0 , Temp= 22.0,Humiity=46.0'
2018-09-30 22:35:52.963660 : b'2018-09-30 12:36:03.585614 ,wlan0 , Temp= 22.0,Humiity=45.0'
2018-09-30 22:35:56.768486 : b'2018-09-30 12:36:07.392491 ,wlan0 , Temp= 22.0,Humiity=45.0'
2018-09-30 22:36:00.755807 : b'2018-09-30 12:36:11.197063 ,wlan0 , Temp= 22.0,Humiity=45.0'
2018-09-30 22:36:04.882768 : b'2018-09-30 12:36:15.184818 ,wlan0 , Temp= 22.0,Humiity=45.0'
2018-09-30 22:36:09.016705 : b'2018-09-30 12:36:19.312541 ,wlan0 , Temp= 22.0,Humiity=45.0'
```

```
2018-09-30 22:39:22.304448 : b'2018-09-30 12:39:32.914541 ,wlan0 , Temp= 71.6f,Humiity=45.0'
2018-09-30 22:39:26.122260 : b'2018-09-30 12:39:36.743870 ,wlan0 , Temp= 71.6f,Humiity=45.0'
2018-09-30 22:39:29.945048 : b'2018-09-30 12:39:40.564242 ,wlan0 , Temp= 71.6f,Humiity=45.0'
2018-09-30 22:39:33.769768 : b'2018-09-30 12:39:44.388045 ,wlan0 , Temp= 71.6f,Humiity=45.0'
2018-09-30 22:39:37.595532 : b'2018-09-30 12:39:48.212044 ,wlan0 , Temp= 71.6f,Humiity=45.0'
2018-09-30 22:39:41.417308 : b'2018-09-30 12:39:52.036053 ,wlan0 , Temp= 71.6f,Humiity=45.0'
```

Figure 8. log.txt file. Before and after reconfiguration

These records show each sensor reading with the corresponding reading time (see Figure 8). The log file on top shows data shows the reading before reconfiguration whereas, the log file on bottom shows the reading after reconfiguration – demonstrating the continuity of service from the IoT node.


ITROLLER/config.xml - pi@192.168.1.11 - Editor - WinSCP
Encoding
Color

```
11</ip>

</nip><nbip>192.168.1.2</nbip><executables><file>TempHumidReaderV1.py</file></
```

TROLLER/config.xml - pi@192.168.1.11 - Editor - WinSCP
Encoding
Color

```
11</ip>

</nip><nbip>192.168.1.2</nbip><executables><file>TempHumidReaderV2.py</file></
```

Figure 9. The configuration file, before and after reconfiguration

The data collection time shown in the log file shows a continuous seamless collection of data, which states that **Requirement 1** is satisfied. A closer look onto the log file will reveal that the time gap between each consecutive record are almost constant. The minor difference can be ignored as it is created due to network lag during data subscription. Neglecting the minor difference, the result concludes that there was no network downtime during reconfiguration stating that **Requirement 2** is satisfied. All the remote API calls are made using the node IP addresses from the configuration file. From this, remote reconfiguration requirement, which is **Requirement 3** is obeyed.

Figure 9 shows the configuration of the node before and after reconfiguration. A closer look into the file reveal that executable program changed its version from v1 to v2. This concludes that software deployment successfully happened during the reconfiguration, thus satisfying **Requirement 4**.

6 Conclusion

Seamless reconfiguration of IOT networks without any network downtime and data loss is one of the major problems faced by engineers for reconfiguring the widely distributed Internet of connected things (IOT) (Xiaohui 2013). The architecture proposed in this paper solves this problem using multiple nodes and the proposed middleware software. The proposed architecture enables seamless collection of data with no network downtime throughout the reconfiguration procedure. All the reconfiguration processes in this network is handled by the proposed middleware software module which is written in python programming language. This middleware module exposes many API interfaces that can be used for reconfiguration of the IOT network.

To check if the architecture has achieved all its goals, a real implementation of this architecture is also created as part of this paper. The results show that the proposed architecture along with the middleware successfully achieved the goal of seamless IOT reconfiguration.

The proposed architecture successfully demonstrated its ability to do seamless reconfiguration of IOT network without any data loss of network downtime. However, this architecture has some limitations and should be improved over time.

This architecture is particularly designed to run the programs which are coded in python programming language. As of now, the developed middleware won't be able to run software written in any other programming language. Overtime, the middleware should automatically identify the software language in which a program is written and should run the program.

The proposed architecture does almost everything without any human interventions by the help of APIs and configuration files. However, the nodes connected to the network is identified using their IP address. As of now, the user must scan the network manually to identify the node IP address and this address must be updated into the configuration files. In future, an automated identification of IP addresses of the nodes can be added into the middleware.

The proposed architecture and middleware system only handle the reconfigurations if the programs are stateless. The proposed architecture would not save the state of the device at any time of the reconfiguration. This is the last limitation of the architecture since the program would lose any running state after the reconfiguration happens. In future, a mechanism to save the program running state and variables must be introduced into the system.

7 References

Becari, W. and Ramirez-Fernandez, F.J., 2016, September. Electrogoniometer sensor with USB connectivity based on the IEEE1451 standard. In *2016 IEEE International Symposium on Consumer Electronics (ISCE)* (pp. 41-42). IEEE.

Beloglazov, A. and Buyya, R., 2010, May. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing* (pp. 826-831). IEEE Computer Society.

Boman, J., Taylor, J. and Ngu, A.H., 2014, October. Flexible IoT middleware for integration of things and applications. In *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing* (pp. 481-488). IEEE.

Chi, Q., Yan, H., Zhang, C., Pang, Z. and Da Xu, L., 2014. A reconfigurable smart sensor interface for industrial WSN in IoT environment. *IEEE transactions on industrial informatics*, *10*(2), pp.1417-1425.

De Prado, A.G., Ortiz, G. and Boubeta-Puig, J., 2017. CARED-SOA: A context-aware event-driven service-oriented Architecture. *IEEE Access*, *5*, pp.4646-4663.

Hauck, S. and Borriello, G., 1997. Pin assignment for multi-FPGA systems. *IEEE transactions on computer-aided design of integrated circuits and systems*, *16*(9), pp.956-964.

Heo, S., Woo, S., Im, J. and Kim, D., 2015, October. IoT-MAP: IoT mashup application platform for the flexible IoT ecosystem. In *2015 5th International Conference on the Internet of Things (IOT)* (pp. 163-170). IEEE.

Iwanicki, K., 2018, July. A distributed systems perspective on industrial IoT. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (pp. 1164-1170). IEEE.

Khajeh-Hosseini, A., Greenwood, D. and Sommerville, I., 2010, July. Cloud migration: A case study of migrating an enterprise it system to iaas. In *2010 IEEE 3rd International Conference on cloud computing* (pp. 450-457). IEEE.

Kodali, R.K. and Soratkal, S., 2016, December. MQTT based home automation system using ESP8266. In *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*(pp. 1-5). IEEE.

Lee, K. and Coulson, G., 2006. Supporting runtime reconfiguration on network processors. *Journal of Interconnection Networks*, *7*(04), pp.475-492.

Lee, K., Paton, N.W., Sakellariou, R., Deelman, E., Fernandes, A.A. and Mehta, G., 2009. Adaptive workflow processing and execution in pegasus. *Concurrency and Computation: Practice and Experience*, *21*(16), pp.1965-1981.

Lee, K., Paton, N.W., Sakellariou, R. and Fernandes, A.A., 2011. Utility functions for adaptively executing concurrent workflows. *Concurrency and Computation: Practice and Experience*, *23*(6), pp.646-666.

Lu, C.H., Hwang, T. and Hwang, I.S., 2016, May. IoT Inventor: A web-enabled composer for building IoT-enabled reconfigurable agentized services. In *2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)* (pp. 1-2). IEEE.

Myint, C.Z., Gopal, L. and Aung, Y.L., 2017, May. Reconfigurable smart water quality monitoring system in IoT environment. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)* (pp. 435-440). IEEE.

Pena, M.D.V., Rodriguez-Andina, J.J. and Manic, M., 2017. The internet of things: The role of reconfigurable platforms. *IEEE Industrial Electronics Magazine*, *11*(3), pp.6-19.

Pazos, N., Müller, M., Aeberli, M. and Ouerhani, N., 2015, December. ConnectOpen-automatic integration of IoT devices. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*(pp. 640-644). IEEE.

Raagaard, M.L., Pop, P., Gutiérrez, M. and Steiner, W., 2017, October. Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing. In *2017 IEEE Fog World Congress (FWC)* (pp. 1-6). IEEE.

Sarray, I., Ressouche, A., Gaffé, D., Tigli, J.Y. and Lavirotte, S., 2015, December. Safe composition in middleware for the internet of things. In *Proceedings of the 2nd Workshop on Middleware for Context-Aware Applications in the IoT* (pp. 7-12). ACM.

Singh, M., Rajan, M.A., Shivraj, V.L. and Balamuralidhar, P., 2015, April. Secure mqtt for internet of things (iot). In *2015 Fifth International Conference on Communication Systems and Network Technologies* (pp. 746-751). IEEE.

Xiaohui, X., 2013, June. Study on security problems and key technologies of the internet of things. In *2013 International conference on computational and information sciences* (pp. 407-410).

Zhou, Z., Du, C., Shu, L., Hancke, G., Niu, J. and Ning, H., 2015. An energy-balanced heuristic for mobile sink scheduling in hybrid WSNs. *IEEE Transactions on Industrial Informatics*, *12*(1), pp.28-40.