Implementation of SCION on IoT Applications

Abhijith Remesh
Email: abhijith.remesh@st.ovgu.de
Manish Ramachandran
Email: manish.rama@st.ovgu.de

***Abstract*— There are several Internet of things (IoT) applications running on legacy networks which are not flexible in terms of path selections, network outages and security concerns. Thus, the current situation demands switching over to a new network platform which is highly scalable, secure, and isolated in terms of its architecture. Thus, a basic IoT application is being attempted which will use Scalable Control and Isolation on Next generation networks (SCION) architecture and hence, surpasses the existing concerns on the current network protocols. The chosen Internet of things (IoT) application is based on Inventory Management wherein immediate actions like stock replenishment or availing discounts can be provided based on the monitored weights of the articles available for sale. The implementation follows a simple client server architecture in SCION wherein one autonomous system (AS) acts as the client and the other autonomous system (AS) acts as the server. The key process includes (I) the client AS generate a request to the server AS to get the current weight data and radio frequency identification (RFID) data (ii) the server AS fetches the respective weight data and the RFID data over a micro-controller and responds back to client AS with the requested data (iii) the data obtained at the client side is visualized using the visualization platform.**

I. INTRODUCTION

THE internet was not designed as a high security network. Security improvements primarily address specific attacks but do not solve the fundamental issues. Scalable Control and Isolation on Next generation networks (SCION) [1] is the first clean-state internet architecture designed to provide route control, failure, isolation and explicit trust information for end to end communication. SCION [1] organizes existing AS into groups of independent routing planes, called isolation domains, which interconnect to provide global connectivity. Isolation domains provide natural isolation of routing failures and misconfigurations, give endpoints strong control for both inbound and outbound traffic, provide meaningful and enforceable trust, and enable scalable routing updates with high path freshness.As a result, the SCION [1] architecture provides strong resilience and security properties as an intrinsic consequence of its design. Besides high security, SCION [1] also provides a scalable routing infrastructure, and high efficiency for packet forwarding. As a path-based architecture, SCION [1] end hosts learn about all the available network path segments, and combine then into end-to-end paths that are carried in packet headers. Thanks to embedded cryptographic mechanisms, path construction is constrained to the route policies of internet service providers (ISPs) and receivers, offering path choice to all the parties: senders, receivers, and ISPs. This approach enables path-aware communication, an emerging trend in networking. These features also enable multi-path communication, which is an important approach for high availability, rapid fail-over in case of network failures, increased end-to-end bandwidth, dynamic traffic optimization, and resilience to Denial-of-Service (DDoS) attacks.

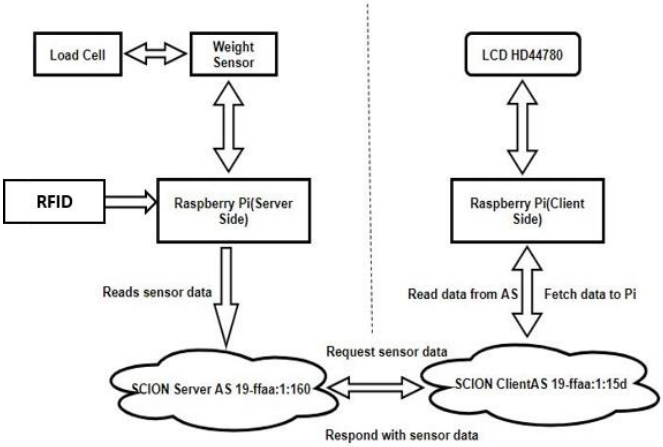

Fig 1. Application schema

THE proposed Internet of things (IoT) application, implemented on the SCION [1] network uses Raspberry Pi 3B+ model as the micro-controller unit and the sensor part consists of two units, one unit comprises of a load cell [4] interfaced with a load cell [4] amplifier HX711 [5] which is essentially a 24 bit high precision analog to digital converter module and the other unit is a MFRC522 [9] radio frequency identification (RFID) module. Both the RFID module and the load cell [4] unit are serially interfaced with the Raspberry Pi. The application also includes a Graphical User Interface (GUI) at the client side displaying the graphical representation of the weight data from the load cell [4] and also the product data from the RFID tag which includes the name, expiry date, unit weight of the product and the capacity of the pallet. The Raspberry Pi end along with the sensor unit forms the SCION [1] server AS whereas the GUI unit forms the SCION [1] client AS. With the GUI unit, the user will have the

opportunity to visualize and monitor the real time weight variations of that specific product which implies insight regarding the sales or use of the product. Based on this insight, the user can take business critical decisions on stock replenishment or discounts generation in real time. An appropriate rigid mechanical assembly is also essential for the correct placement of load cell [4] below the pallet so that the product can be mounted on the pallet in a stable manner. Apart from this, a Liquid Crystal Display (LCD) interface is also enabled at another Raspberry Pi end which displays the weight data over the network and is not under the scope of this report and the implementation regarding the same is discussed in the other report.

A. Overview

The section II discusses about the installation and configuration of SCION [1] network on the personal computer and Raspberry Pi respectively and the section III discusses in detail about the hardware devices used which includes the micro-controller, the sensor modules and how they are interfaced with each other and also, the configuration of the load cell [4] unit and the RFID module. The section IV describes the configuration and implementation of the SCION [1] client-server and the flow of communication between them and the section V describes the implementation of the Graphical User Interface using Node-Red [6] at the client side. The next section VI provides the results of the application with regards to the accuracy and robustness. After this, the section VII describes the conclusion and possibilities of further developments. Then it is followed by the reference section and then at last, the appendix section where the complete implementation scripts can be found.

II. CONFIGURING SCION

THIS chapter discusses how the SCION [1] environment is established using a virtual machine on the personal computer which acts as the client and how it is being enabled on a Raspberry Pi, transforming it into a stand alone SCION [1] AS enabled Internet of things (IoT) device which acts as the server.

The entire IoT application consists of three SCION [1] AS node instances, (i) one SCION [1] AS node configured as a server on one Raspberry Pi [2] which is interfaced with the weight sensor, (ii) another SCION [1] AS node configured as a client on another Raspberry Pi [2] which is interfaced with the LCD unit where the weight data is displayed (not within the scope of this report), (iii) another SCION [1] AS node configured as a client on personal computer using virtual machine where a GUI is implemented for visualization purposes with Node-Red [6].

A. Installation of SCION Virtual Machine

A new SCION [1] Lab AS is generated with the a unique id of the format: AS 19-ffaa:x:xxx with it's attachment point as 19-ffaa:0:1303 (OVGU) and port as 5000 which is to be installed inside a virtual machine (Virtual box).The Virtual machine (VM) configuration file of this generated AS node is then downloaded. This SCION [1] AS configuration file is available for download at the site www.scionlab.org where registration of a user account is mandatory. Then the downloaded compressed configuration file is extracted accordingly which is followed by the setting up and installation of virtual environment using Virtual Box and Vagrant. After this, certain commands are executed in the command prompt to run and enter the virtual machine after which we navigate to the path where the scion AS is installed and run certain commands to start the SCION [1] network. Upon successful execution, we shall be able to receive beacons from other AS nodes in the SCION [1] network with execution of certain commands and also, we shall be able to get the multiple paths available between the nodes by executing another set of specific commands. The reception of beacons implies that the SCION [1] network is correctly established and our SCION [1] AS node is active. The detailed explanation with all the relevant commands about the installation of SCION [1] on virtual machine is explained in SCION lab tutorials available at www.scionlab.org As already mentioned, this SCION [1] AS node installed within the virtual machine acts as the client of our IoT application where the GUI will be implemented.

B. Installation of SCION on Raspberry Pi

Another new SCION [1] Lab AS is again generated with the another unique id of the format: AS 19-ffaa:x:xxx with it's attachment point as 19-ffaa:0:1303 (OVGU) and port as 5000. Despite using the Virtual Machine configuration, since our requirement is to install the SCION [1] AS on a dedicated device, (Raspberry Pi [2]) the respective option "*Install on a dedicated SCION system"* is selected instead of considering the option "*Install inside a virtual machine*". Thus the configuration file of this generated AS node is then downloaded in a compressed format. As the appropriate SCION [1] image file was not available for the recent version of Raspberry Pi [2] 3B+ model, only two options were left, either to manually install SCION [1] on the top of Raspbian [10] operating system or to install SCION [1] on the Ubuntu Mate for Raspberry Pi [2] 3 B+ model. Since there were some issues being encountered with the former option, the latter option was taken into consideration for the SCION [1] installation and further implementation and the same is explained as follows:

- Download a copy of Raspbian OS for Pi 3B+ and Ubuntu MATE for raspberry Pi 3B.
- Copy and then replace the following files from Raspbian [10] OS to Ubuntu MATE: *bootcode.bin,*

fixup.dat, start.elf, bcm270-rpi-3-b-plus.dtb, kernel17.img, all contents from */lib/modules/4.9.80-v7+* and */lib/firmware/brcm/*.

- Flash the new OS to the SD card via Etcher and perform the similar steps for Ubuntu MATE as described on www.scionlab.org.

Instead of command prompt, Putty has been used to connect with the Raspberry Pi and then to run and host the SCION [1] network from the Raspberry Pi [2]. Thus, the SCION [1] network is established on the IoT device which acts as the SCION [1] server AS for the proposed IoT application.

III. HARDWARE DEVICES IN CONCERN

This section explains in detail about the several hardware units used in the application which includes the following, (i) Raspberry Pi [2] which acts as the primary micro-controller unit, (ii) Load cell [4] of weight range up to one kg, (iii) load cell [4] amplifier HX711 [5] and (iv), MFRC522 [9] RFID reader module and explains how these devices are interfaced with each other to form a stand alone integrated IoT sensor device which transmits product data (weight data and RFID data) over the SCION [1] network.

A. Device Specifications

- Raspberry Pi 3B + model

The application uses Raspberry Pi [2] as the micro-controller unit and around two Raspberry Pi [2] has been used upon which two separate SCION [1] AS nodes are installed respectively, one acts as the server interfaced with the RFID module and load [4] cell and the other acts as the client interfaced with the LCD unit where the weight data is being displayed (not within the scope of this report).

- Load cell

As the main objective of application is to monitor and fetch the real time weight, load cell [4] of weight range up to 1 kg has been used. It is essentially a transducer that detects an applied load (force weight) and then changes it into an electrical signal output that is proportional to the load. The load causes a deformation in the dimensions of the underlying strain gauge; this deformation changes the electrical resistance which further changes the output electrical voltage as the wheat stone bridge is not balanced. The Load cell [4] consists of four pin outlets: red, black, green and white.

- Load cell amplifier HX711

The voltage signals which comes as output from the load cell [4] is then amplified with a HX711 [5] , a 24 bit analog to digital converter. This amplifier has got four pin outlets namely E+ (excitation) or VCC, E- (excitation) or ground and two outputs, that is A- and A+.

- MFRC522 RFID reader

The MFRC522 [9] is a highly integrated reader/writer for contact-less communication. It uses the concept of radio-frequency identification and electromagnetic fields to transfer data over short distances. RFID employed in our project is useful to identify product data for further processing and analyzing. An RFID system uses RFID tags are attached to the object which is to be identified. Key chain and an electromagnetic card has been used in our project which has their own unique identification ids. It has got the following pins which are being utilized namely SDA, SCK, MOSI, MISO, GND, RST.

B. Interfacing the Load Cell and HX711 amplifier with Raspberry pi

The HX711 [5] weight amplifier is interfaced with the load cell [4] using it's following connections

- Excitation (E+) or VCC is red
- Excitation (E-) or ground is black
- Output (A+) is white
- Output (A-) is green

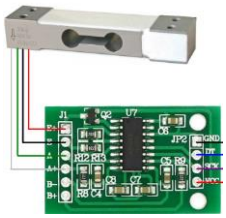

Fig 1. Load cell-HX711 connection

This load cell HX711 [5] integrated sensor unit is then interfaced with the Raspberry Pi serially as follows:

- Vcc of HX711 to Pin 2 (5V)
- GND to Pin 6 (GND)
- DT to Pin 29 (GPIO 5)
- SCK to Pin 31 (GPIO 6)

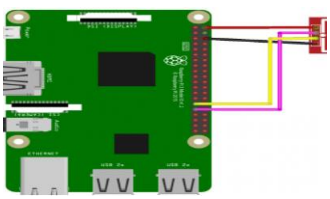

Fig 2. Raspberry Pi-HX711-load cell connection

This sensor unit comprising of load cell [4] and HX711 [5] amplifier needs to be tested and calibrated accordingly to ensure accurate weight readings after which real time weight data is acquired. Several public libraries has been used to test the load cell [4] wherein raw voltage values would increase and decrease as weights are added and removed respectively and also to calibrate the load cell [4] unit wherein known weight values are used to set the appropriate reference values. Once the reference values are identified, these values are then hard coded in the main real time weight acquisition code logic *Weight.go* where weights of unknown objects can be obtained and observed in the console.

C. Interfacing the MFRC522 reader with Raspberry Pi

RFID tags are attached to the product or pallet which is to be identified. The application primarily uses a card and a key chain as the RFID tag. Each tag is associated with a unique id. The RFID reader sends a signal to the tag and read it's response. There are 8 hardware connections for RFID sensor with the Raspberry Pi [2] as follows:

- SDA connects to Pin 24
- SCK connects to Pin 23
- MOSI connects to Pin 19
- MISO connects to Pin 21
- GND connects to Pin 6
- RST connects to Pin 22
- 3.3v connects to Pin 1

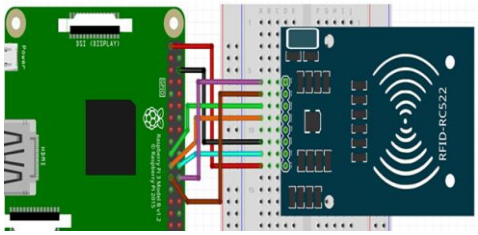

Fig 3. Raspberry Pi-MFRC522 reader module

Certain configuration steps needs to be carried out for their interface. By default the Pi has the SPI (Serial Peripheral Interface) disabled, which is a prerequisite for the RFID reader to function and therefore, needs to be enabled using rasp i-config tool as follows:

- Run the command "s*udo raspi-config*"
- Select "*Interfacing options*"
- Select "*P4 SPI*" and then, select "*Yes*"
- Run the command "*sudo reboot*" to reboot the Pi
- Run the command ""lsmod | grep spi" to check.
- And ensure if spi_bcm2835 is listed.

Python scripts has been employed to interact with the module mainly for two purposes, first is to read the default unique identification (UID) from the detected RFID tag and to send the data packet as JSON object over a TCP server. A library which is an implementation of the MFRC522 [9] interface has been cloned, which also handles all the heavy lifting for talking with the RFID over the Raspberry Pi [2] SPI interface. The Read.py script is designed with the cloned libraries to communicate with RFID and obtain the UID (An RFID card has a UID which is a unique number associated with the tag, similar to a serial number) of each tag whenever it is swiped with RFID. On running the script, The card's UID will be displayed and printed on the command terminal. This UID is noted for further processes. A Transmission Control Protocol (TCP) socket server is also implemented in the python script so that the server responds back with the UID data to respective client requests. A TCP Client go script has been also developed for testing the client server communication. As a result, the Python script performs two functionalities in it's code, one is to read the UID of the RFID tags if detected and then, to host a TCP server which would expose the data packet over a specific port. With this, whenever any client makes a request to this port, the server responds back with the data packet in Javascript Object Notation (JSON) format containing the UID. The complete code can found in the appendix section.

IV. CONFIGURING THE SCION SERVER-CLIENT

This chapter discusses about the establishment of client server communication over the SCION [1] network. The sequence of operation is as follows:

- Initially, the SCION [1] AS node installed on the Raspberry Pi [2] initiated by specifying a particular port number and respective detected Internet Protocol address (IP) of the Raspberry Pi [2]. This acts as the server of our application.
- Subsequently, the SCION [1] AS node installed within the virtual machine on personal computer is started by specifying a particular port number and respective IP address of the personal computer.
- As a result, the client server communication is enabled and the client should be able to receive the responses from the server.

The basic implementation of the above discussed process is explained below by setting up a basic SCION [1] server and client transmitting raw data packets between them. This is done for testing purposes.

- A basic SCION [1] server go script is developed which send sample data packets to SCION [1] client. By

running the below command as below, the SCION [1] server gets started.

"*go run Scion_server.go -s 19-ffaa:1:161,[192.168.137.185]:30102*".

- A basic SCION [1] client go script is developed which obtains the data packets from SCION[1] server. By running the below command as below, the SCION [1] client starts.

"*go run Scion_client.go -s 19-ffaa:1:161,[192.168.137.185]:30102*".

Validation and Verification of the communication is done by the successful reception of the raw data values at the client SCION [1] AS node.

A. Configuring the SCION client

This chapter discusses the slight modification in the SCION [1] client go script to receive the JSON object encapsulating product data, weight values and UID from the hosted SCION [1] server AS node. Apart from that, the script also performs the hosting of a Hyper Text Transfer Protocol (HTTP) socket which is utilized for exposing the JSON object over a HTTP connection so that the visualization tool Node-Red [6] can use a http get request to access the JSON object and then visualize the same. The script is executed by running the below command as shown.

- "*go run Scion_client.go -c 19-ffaa:1:bfa,[192.168.1.130]:30102 -s 19-ffaa:1:161,[192.168.137.185]:30102* "

Where "*30102"* indicates the port number,"*19-ffaa:1:bfa,"* indicates the SCION [1] client AS node, "*[192.168.1.130]"* indicates the client IP address(Personal Computer), "*19-ffaa:1:161*" indicates the SCION [1] server AS node and "*[192.168.137.185]"* indicates the server IP address(Raspberry Pi [2]). The complete script can be found in the appendix section.

B. Configuring the SCION server

This section discusses the integration of the weight acquisition go script with the basic SCION [1] server go script developed earlier. As a result, the final SCION [1] server go script "*Weight_server_full_rv2*.go" performs the below functionalities.

- Act as a TCP Socket client and fetch UID values from Python TCP socket server whenever the card is swiped with RFID sensor.
- Fetch appropriate weight values from load cell [4]-HX711 [5] sensor.
- The product data like name, expiry date, capacity and so on are also hard coded for each UID in the SCION [1] server go script.
- Amalgamate product data associated with respective UID and their corresponding weight as one compact JSON object and send it to requesting client SCION [1] AS node.

This script upon execution produces a JSON object with product UID, product data and the respective real time weight readings at the SCION [1] client AS node. The script is executed by running the below command as shown.

- "*go run weight_server_full.go -s 19-ffaa:1:161,[192.168.137.185]:30102*"

Where "*30102"* indicates the port number, "*[192.168.137.185]*" is the detected dynamic IP of the Raspberry Pi [2] and "*19-ffaa:1:161*" is the SCION [1] AS node installed on the Pi. The complete script can be found in the appendix section.

C. Client Server Communication flow

THIS chapter discuses the chronological sequence of data flow when a request is made from the client SCION [1] AS end that is, from the Node Red dashboard end.

The below block diagram depicts the complete components involved in the scope of application. The part (A) is within the scope of this report and the part (B) corresponds to the other report. When a request is initiated from the client SCION [1] AS node, the chronological order of data flow is as follows:

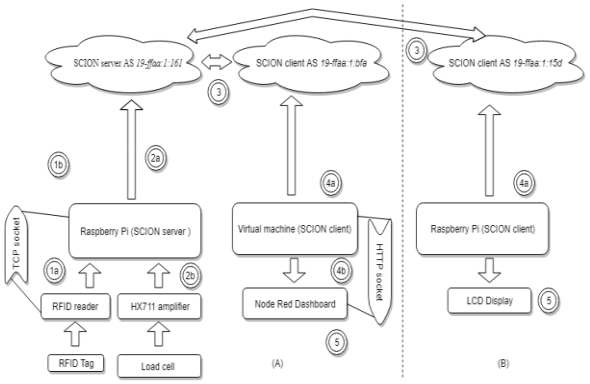

Fig.4:Client-server data flow

- The Raspberry Pi [2] initiates the RFID reader to read the UID of the scanned RFID tag if detected.
- The Raspberry Pi [2] hosts a TCP socket server bound to a specific port. The data packet containing the UID is converted to JSON format and then send as bytes to this TCP connection and starts listening for any incoming

request. Whenever this TCP server receives a request, it responds with this data packet.

- Then, the Raspberry Pi [2] runs and connects to the SCION [1] network as a SCION [1] AS server node by identifying the SCION [1] server address, scion path and dispatcher path. Once identified, the SCION [1] server starts listening over a UDP connection for any incoming requests. Then a TCP connection is established with that specifc port by the SCION [1] server and the HX711 [5] sensor is also initialized. Whenever any incoming request is encountered from the SCION [1] AS client node, it sends a sample data packet as request to the TCP socket over a TCP connection. It gets back the UID as the TCP response from the TCP server.
- If the UID obtained is a valid existing one, then the real time weight reading from the HX711[5]-load cell [4] unit is fetched and then appended to the data packet. Moreover the UID, current number, current time are also appended to the data packet which is then converted into JSON format.
- This data packet is then sent to the respective client SCION[1] AS node over the SCION [1] network.
- The client SCION [1] AS node now receives the data packet as a JSON object encapsulating the UID, product name, product expiry date, current time, unit weight, capacity, current number and weight.
- The client SCION [1] AS hosts a HTTP socket on a specific port and establish a HTTP connection and starts listening. Then, it exposes this data packet as JSON object over this defined port. Whenever, this HTTP server encounters a request from any browser or user, it sends with this JSON object as response.
- The visualization tool, Node-Red [6] dashboard makes a HTTP GET request to this defined port, and gets this JSON object as response which is then processed, parsed and displayed using several node functions available in the Node-Red [6].

The other section is not discussed in this report as it is not within the scope of our application. This is how the data flow happens between the SCION [1] server AS node and the SCION [1] client AS node. The SCION [1] server AS node acts as a client with respect to the TCP server and it acts as a server for the [1] client AS node at the same node. Similarly, the SCION [1] client AS node acts as the client with respect to the SCION [1] server AS node and acts as a server for the Node-Red [6] dashboard simultaneously. It can be said that both the SCION [1] server and SCION [1] client AS node perform dual roles, that of a client and server.

V. IMPLEMENTATION OF GUI

THIS chapter explains how the graphical user interface has been implemented at the client SCION [1] AS node. The flow-based, visual programming development tool, Node-Red [6] has been employed for this purpose. Node-Red [6] requires data in the JSON Object format so that the desired flow can be developed using nodes which is why, the data packets are transferred over the SCION [1] network as JSON objects. The tool makes use of a "http" input node which performs a HTTP GET function to access the JSON object which is why, the SCION [1] client go script has been modified to incorporate a HTTP socket server being hosted at a specific port. The data packet, in particular the JSON object being received at the client SCION [1] AS node is exposed over a HTTP socket server which is then accessed by the Node-Red [6] HTTP GET input function node. This JSON Object is then parsed into different and separate data using the other node functions available on the Node-Red [6] platform. The different main node functions used for the visualization are as follows.

- A "*http*" input type node is used to access the JSON object (data packet) hosted on a HTTP server at port *4000*. The node is configured accordingly to use a *GET* method at the Uniform Resource Locator (URL) (port 4000). The JSON object is received as a message payload at the Node-Red [6] end.
- Several change type nodes are used to parse several data information from the entire message payload (JSON object). Change type nodes are used to process and parse the temperature, humidity, product name, expiry date, capacity, current number, weight information from the message payload.
- Several dashboard nodes like "*text*", "*gauge*", "*chart*" are used to visualize all these parsed separate information as a gauge, wave, level default text types of visualization.

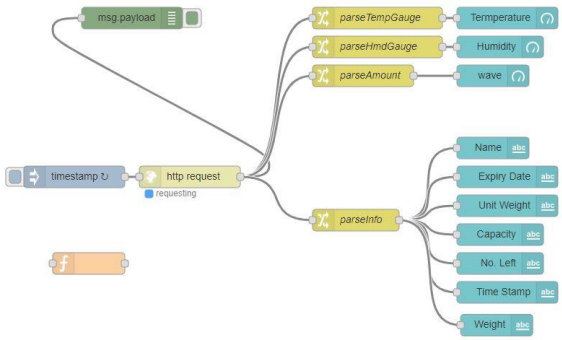

Fig.5:Node-red flow based diagram

This acts as a Graphical User Interface for the users so that they can monitor the inventory status of products in real time which eliminates the need to do frequent physical inventory inspection and take business decisions accordingly. This also facilitates the users to take business decisions either to replenish the decreasing stock or to grant discounts in real time.

VI. RESULTS

THIS chapter describes the results observed after the completion of relevant development tasks during the entire course of the application development which involves the following:

- Rigid assembly with the appropriate placement of the load cell [4], RFID reader module and the Raspberry Pi [2] to form a stand-alone "*smart pallet"* unit. The assembly comprises of a mounting plate which acts as the platform for keeping the RFID tag attached product pallet and a placeholder unit which contains the RFID, load cell [4] connected Raspberry Pi[2].

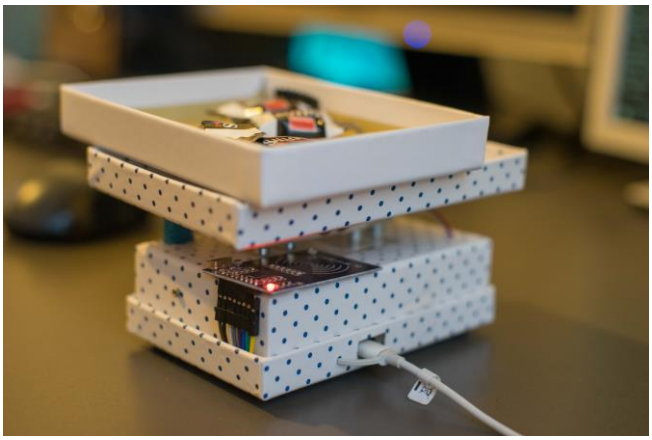

Fig.6:Smart pallet assembly

- Reception of raw weight readings from the load cell [4]-Hx711 [5] sensor unit which changes proportionally with the applied load.

Fig.7:

Fig.7:Raw weight readings

- Reception of the JSON object at the client SCION [1] AS node wherein it hosts over a HTTP socket on port 4000 which contains the UID of the scanned tag, the product name, weight reading, product expiry date, pallet capacity and so on. The weight readings are dynamics and it keeps on varying as per the applied load.

Fig.8:JSON data at port 4000

- Visualization of the product details, especially the real time weight readings on the Node-Red [6] browser platform. The Dashboard visualization in the form of gauges, level represents the changes in weight graphically in real time.

Fig.9:Node-Red dashboard visualization

VII. CONCLUSION AND FURTHER WORK

This scientific work focuses on the implementation of an IoT application over the SCION [1] network. The scope of the scientific work in general involves the establishment of a client server communication on the SCION [1] network by employing one SCION [1] AS node as the server and the other SCION [1]AS node as the client on an IoT application context. We came up the concept of real time inventory management of products as an IoT use case which is then deployed over the next generation network architecture, SCION [1]. The application which we have realized is just an initial development and posses wide scope of developmental progress in several application contexts. This application can find it's usage in wholesale, retail, manufacturing, warehouses and logistics domains wherever an inventory status and management of products are required on it's further development like using load cell [4] of wide range. The application can extend it's functionalities to automatic

discount generation in real time, automatic order generation for stock replenishment in real time, monitor the sales data of the product which can further used for reconciliation with the point of sales report. All these scopes could been explored if there were no time constraints.

Another possible extension is to incorporate other sensors like temperature, humidity and pressure sensors so which would make the use case more realistic and even applicable to monitoring and transportation of cold storage and fresh food products where the real time temperature, pressure and humidity of the environment can be known and hence the temperature of the cold storage area can be remotely controlled and changed accordingly. The same use case also applies to the monitoring of chemicals at chemical industries.

As a conclusion, the existing application has basically three entities (I) one being the SCION [1] server entity at the pallet integrated with Raspberry Pi [2], RFID reader, load cell [4], HX711 [5] amplifier and then, the SCION [1] client entity at the pallet end integrated with another Raspberry Pi [2] and LCD so that product details are displayed on the LCD and at last, another SCION [1] client entity at the user side, where the product details are monitored in real time. The application can be imagined to be extended such that products stored in all pallets can be interfaced on a single SCION [1] AS server configured Raspberry Pi [2] using some switching mechanism and then, these product data are then transferred over the SCION [1] network on to the multiple SCION [1] client AS at remote locations. The provision of updating and rewriting the RFID tag details attached to each of the pallet can also be considered as a further possible extension.

REFERENCES

[1] SCION lab Coordination Service, SCION is the first clean-slate Internet architecture designed to provide route control, failure isolation, and explicit trust information for end-to-end communication. Available: https://www.scionlab.org

[2] Raspberry Pi - A small and affordable computer that you can use to learn programming. Available: https://www.raspberrypi.org/

[3] balenaEtcher, An open source project by balena. Available: https://www.balena.io/etcher/

[4] Micro load cell data sheet
Available: https://www.robotshop.com/media/files/pdf/datasheet3133.pdf

[5] HX711, 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales. Available:
https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

[6] Node-Red, Flow-based programming for Internet of things. Available: https://nodered.org/

[7]Go, The Go Programming Language. Available: https://golang.org/

[8] Python, Python is a programming language that lets you work quickly and integrate systems more effectively. Available: https://www.python.org/

[9]MFRC522 Standard performance MIFARE and NTAG front end. Available: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[10] Raspbian, Raspbian is the Foundation's official supported operating system. Available: https://www.raspberrypi.org/downloads/raspbian/

[11] docker, Build, Ship, and Run Any App, Anywhere. Learn about the only enterprise-ready container platform to cost-effectively build and manage your application Available :https://www.docker.com/

[12] Wireshark, Wireshark is the world's foremost and widely-used network protocol analyzer. Available: https://www.wireshark.org/

[13] SCION tutorials, Welcome to SCION Tutorials. Available: https://netsec-ethz.github.io/scion-tutorials/

[14] Ubuntu Mate for Raspberry Pi 2 and 3, Available: https://ubuntu-mate.org/raspberry-pi/

[15] Github, GitHub brings together the world's largest community of developers to discover, share, and build better software. Available: https://github.com/

[16] HX711-Go library, MichaelS11
Available : https://github.com/MichaelS11/go-hx711

[17] SCION Browser AS Visualizations. Available: https://netsec-ethz.github.io/scion-tutorials/as_visualization/browser_asviz/

[18] SCION bwtester application. Available :https://netsec-ethz.github.io/scion-tutorials/sample_projects/bwtester/

[19] Network Security Group at ETH Zürich. Scion-apps. Available: https://github.com/netsec-ethz/scion-apps

ABBRREVIATIONS

SCION	Scalability, Control , And Isolation On Next Generation Networks
IoT	Internet Of Things
RFID	Radio Frequency Identification
AS	Autonomous System
ISP	Internet Service Provider
DDoS	Denial of Service
GUI	Graphical User Interface
LCD	Liquid Crystal Display
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
JSON	Javascript Object Notation
IP	Internet Protocol
HTTP	Hyper Text Transfer Protocol

APPENDIX

(a)RFID_TCPserver.Py

```
import RPi.GPIO as GPIO
import MFRC522
import signal
from multiprocessing import Process, Value, Array
import Adafruit_DHT
import socket
import time
import json


continue_reading = True
sensor = Adafruit_DHT.DHT11
pin = 17
MIFAREReader = MFRC522.MFRC522()
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.bind(('0.0.0.0',10000))
sock.listen(1)

def readTempAndHumidity(arr):
while continue_reading:
arr[1], arr[0] = Adafruit_DHT.read_retry(sensor, pin)
# if arr[0] is not None and arr[1] is not None:
#     print('Temp={0:0.1f}*
Humidity={1:0.1f}%'.format(arr[0],arr[1]))
# else:
#     print('Failed to get reading. Try again!')
# if continue_reading == False:
#     return
time.sleep(2)

# Capture SIGINT for cleanup when the script is aborted
def end_read(signal,frame):
global continue_reading
# print "Ctrl+C captured, ending read."
continue_reading = False
sock.close()
#TempReadProcess.join()
GPIO.cleanup()

# Hook the SIGINT
signal.signal(signal.SIGINT, end_read)

arr = Array('d',(0.0,0.0))

TempReadProcess = Process(target=readTempAndHumidity,
args=(arr,))
TempReadProcess.start()


uid_str = "[]"
while True:
c,a = sock.accept()

while True:
data = c.recv(128)
temperature = int(arr[0])
humidity = int(arr[1])

if continue_reading:

# Scan for cards
(status,TagType) =
MIFAREReader.MFRC522_Request(MIFAREReader.PICC_
REQIDL)

# If a card is found

(status,uid) = MIFAREReader.MFRC522_Anticoll()
if status == MIFAREReader.MI_OK:
uid_str = str(uid)

(status,TagType) =
MIFAREReader.MFRC522_Request(MIFAREReader.PICC_
REQIDL)


data = {"UID" : str(uid),
"Temperature" : temperature,
"Humidity" : humidity}
print(str(data))
data = json.dumps(data)
c.send(bytes(data))
if not data:
c.close()
break
```

(b)weight_server_full_rv2.go

```
package main

import (
//"bufio"
"encoding/json"
"flag"
"fmt"
"log"
"net"
"time"

//"os"
//"strings"

"sync"

"github.com/MichaelS11/go-hx711"

"github.com/scionproto/scion/go/lib/sciond"
"github.com/scionproto/scion/go/lib/snet"
)

type Rfidth struct {
Humidity    int
UID         string
```

```
        Temperature int
    }

type Message struct {
        Name       string
        Exp        string
        Time       string
        Unitweight float64
        Capacity   int
        CurrentNo  int
        TempL      int
        TempH      int
        HmdL       int
        HmdH       int
        Temp       int
        Hmd        int
        Wght       int
        UID        string
    }

func check(e error) {
        if e != nil {
            log.Fatal(e)
        }
    }

var weightData map[string]string
var weightDataLock sync.Mutex

    func init() {
    weightData = make(map[string]string)
    }

// Obtains input from weight observation application

    func printUsage() {
    fmt.Println("weightserver -s ServerSCIONAddress")
    fmt.Println("The SCION address is specified as ISD-
        AS,[IP Address]:Port")
    fmt.Println("Example SCION address 17-
        ffaa:0:1102,[192.33.93.173]:42002")
    }

    func main() {

        var (
        serverAddress  string
        sciondPath     string
        sciondFromIA   bool
        dispatcherPath string

        err    error
        server *snet.Addr

        udpConnection snet.Conn
        )

var msg = []Message{Message{"Sallos",
"10/10/2020", "", 5.4, 21, 0, 0, 40, 10, 90, 20, 30, 0, "[249, 20,
56, 86, 131]"},
        Message{"Toffee", "20/10/2020", "", 4, 50,
0, 0, 50, 10, 90, 20, 30, 0, "[41, 106, 118, 72, 125]"}}
    var noPallette = Message{"NoPallete", "", "", 0, 1, 0,
        0, 50, 0, 100, 0, 0, 0, "[]"}
        var sendData Message
        sendData = noPallette
    var rfth = Rfidth{UID: "[]"}
    // Fetch arguments from command line
    flag.StringVar(&serverAddress, "s", "", "Server
        SCION Address")
    flag.StringVar(&sciondPath, "sciond", "", "Path to
        sciond socket")
    flag.BoolVar(&sciondFromIA, "sciondFromIA",
false, "SCIOND socket path from IA address:ISD-AS")
    flag.StringVar(&dispatcherPath, "dispatcher",
        "/run/shm/dispatcher/default.sock",
            "Path to dispatcher socket")
        flag.Parse()

    // Create the SCION UDP socket
        if len(serverAddress) > 0 {
            server, err =
snet.AddrFromString(serverAddress)
            check(err)
        } else {
            printUsage()
    check(fmt.Errorf("Error, server address
needs to be specified with -s"))
        }

    if sciondFromIA {
        if sciondPath != "" {
        log.Fatal("Only one of -sciond or -
sciondFromIA can be specified")
        }
        sciondPath =
sciond.GetDefaultSCIONDPath(&server.IA)
    } else if sciondPath == "" {
        sciondPath =
sciond.GetDefaultSCIONDPath(nil)
    }
    snet.Init(server.IA, sciondPath, dispatcherPath)
    udpConnection, err = snet.ListenSCION("udp4",
        server)
        check(err)

    defer udpConnection.Close()

    err = hx711.HostInit()

hx711, err := hx711.NewHx711("GPIO6", "GPIO5")
        check(err)
    // SetGain default is 128
    // Gain of 128 or 64 is input channel A, gain of 32 is
        input channel B
    // hx711.SetGain(128)
    // make sure to use your values from calibration
        above
    hx711.AdjustZero = -72782
    hx711.AdjustScale = -1960
```

```
tcpreq := "hello from client"

tcpAddr, err := net.ResolveTCPAddr("tcp",
"0.0.0.0:10000")
if err != nil {
panic(err)
}
conn, err := net.DialTCP("tcp", nil, tcpAddr)
defer conn.Close()
if err != nil {
panic(err)
}

receivePacketBuffer := make([]byte, 2500)
sendPacketBuffer := make([]byte, 3000)
tcpReply := make([]byte, 1024)

for {
_, clientAddress, err :=
udpConnection.ReadFrom(receivePacketBuffer)
check(err)

// Packet received, send back response to
same client

//time.Sleep(200 * time.Microsecond)
conn.Write([]byte(tcpreq))

n, err := conn.Read(tcpReply)
check(err)
err = json.Unmarshal(tcpReply[:n], &rfth)
check(err)
if rfth.UID != "[]" {
for i := 0; i < len(msg); i++ {
if string(rfth.UID) ==
(msg[i].UID) {
sendData =
msg[i]
break
}
}
}
data, err := hx711.ReadDataMedian(11)
check(err)

if string(sendData.UID) != "[]" {
sendData.CurrentNo = int(data /
sendData.Unitweight)
sendData.Time =
string(time.Now().Format("Jan 2 15:04:05"))
} else {
sendData = noPallette
}
sendData.Temp = rfth.Temperature
sendData.Hmd = rfth.Humidity
sendData.Wght = int(data)
b, _ := json.Marshal(sendData)
fmt.Println(string(b))
copy(sendPacketBuffer, b)

for i := 0; i < 3; i++ {
_, err =
udpConnection.WriteTo(sendPacketBuffer[:len(b)],
clientAddress)
if err == nil {
break
} else {
log.Println(err)
}
}
check(err)
sendData = noPallette
}

}
```

(c) weight_client_retry.go

```
package main

import (
"flag"
"fmt"
"log"
"net/http"

"github.com/scionproto/scion/go/lib/sciond"
"github.com/scionproto/scion/go/lib/snet"
)

func check(e error) {
if e != nil {
log.Fatal(e)
}
}

func printUsage() {
fmt.Println("scion-sensor-server -s
ServerSCIONAddress -c ClientSCIONAddress")
fmt.Println("The SCION address is specified as ISD-
AS,[IP Address]:Port")
fmt.Println("Example SCION address 1-
1,[127.0.0.1]:42002")
}

var (
clientAddress  string
serverAddress  string
sciondPath     string
sciondFromIA   bool
dispatcherPath string
udpConnection  snet.Conn
err            error
local          *snet.Addr
remote         *snet.Addr
)
```

```
func main() {

    // Fetch arguments from command line
    flag.StringVar(&clientAddress, "c", "", "Client SCION Address")
    flag.StringVar(&serverAddress, "s", "", "Server SCION Address")
    flag.StringVar(&sciondPath, "sciond", "", "Path to sciond socket")
    flag.BoolVar(&sciondFromIA, "sciondFromIA", false, "SCIOND socket path from IA address:ISD-AS")
    flag.StringVar(&dispatcherPath, "dispatcher", "/run/shm/dispatcher/default.sock", "Path to dispatcher socket")
    flag.Parse()

    // Create the SCION UDP socket
    if len(clientAddress) > 0 {
        local, err = snet.AddrFromString(clientAddress)
        check(err)
    } else {
        printUsage()
        check(fmt.Errorf("Error, client address needs to be specified with -c"))
    }
    if len(serverAddress) > 0 {
        remote, err = snet.AddrFromString(serverAddress)
        check(err)
    } else {
        printUsage()
        check(fmt.Errorf("Error, server address needs to be specified with -s"))
    }

    if sciondFromIA {
        if sciondPath != "" {
            log.Fatal("Only one of -sciond or -sciondFromIA can be specified")
        }
        sciondPath = sciond.GetDefaultSCIONDPath(&local.IA)
    } else if sciondPath == "" {
        sciondPath = sciond.GetDefaultSCIONDPath(nil)
    }
    snet.Init(local.IA, sciondPath, dispatcherPath)
    udpConnection, err = snet.DialSCION("udp4", local, remote)
    check(err)

    http.HandleFunc("/", dataHandler)

    http.ListenAndServe(":4000", nil)

}

func dataHandler(w http.ResponseWriter, r *http.Request) {
    receivePacketBuffer := make([]byte, 2500)
    sendPacketBuffer := make([]byte, 0)
    n := 0
    for i := 0; i < 3; i++ {
        n, err = udpConnection.Write(sendPacketBuffer)
        check(err)

        n, _, err = udpConnection.ReadFrom(receivePacketBuffer)
        if err == nil {
            break
        } else {
            log.Println(err)
        }
    }

    check(err)

    fmt.Println(string(receivePacketBuffer[:n]))

    w.Write(receivePacketBuffer[:n])
}
```

(d)SCION Raspberry installation method