

SDN IMPLEMENTATION OF CLOUD AND IOT BASED LIQUID SPILL AND LEAKAGE DETECTION SYSTEM

Deepak Khairwal

Abstract

The increase in trade among nations specially of liquid commodity (crude oil, diesel, other chemicals) have come with an increase in incidents of fault leakages in the containers of the vessels carrying the goods resulting in large oil spills, illegal oil tapping cases in recent times have further worsened the situation of oil produces with pipeline rupturing cases adding to the demand of an automated system which could sense the proximity environment and report to the concerned authorities about an alarming situation. One such feasible and effective solution involves integration of various technologies. SDN advocates shifting the control plane to a remote centralized device or application called as controller. The controller is responsible for selecting the shortest routing path to forward the packets using the Optimal path selection algorithm .The coupling of data and control plane on the same device makes it cumbersome to develop new applications and to modify the existing network configuration . SDN has been proved to improve the manageability and efficiency of the network in recent times. The ever increasing internet devices are forcing IoT applications to use cloud services more and more, thus integration IoT and cloud is far or less demanding. Thus there is a requirement of a middleware which could provide compatibility for both platforms. The Openflow protocol allows a network to control the IoT devices employed in multi-tenant and cloud based deployments. The WSN sensors need to read analog values continuously and autonomously for long time of periods. Its impractical to replace batteries in those sensors and some hostile environment makes it impossible. The energy saving schemes must be application specific rather than cost oriented. This paper highlights the steps involved in creating a virtual software defined network using a network emulator called mininet. A real time experimental setup using Arduino board and IoT sensors which senses any leakage or spill and writes the data to the cloud which is eventually read by the virtual hosts in the SDN. This is followed by evaluation of various network topologies and energy-saving techniques in our wireless sensor network.

Keywords: *Software defined networking(SDN); Mininet; OpenFlow; OpenDayLight controller; Thingspeak cloud; Arduino; ESP8266.*

1. Introduction

Reported incidents of cargo ships spilling oil and faulty pipeline leakages have pressed the need for an automated liquid spill and leakage detection system. Oil tapping incidents not only affects the producers but the environment as in the case of 29th Jan 2019 in Mexico city where a blast in pipeline followed an illegal tapping to steal liquid fuel [25]. The solution requirements are difficult to be satisfied by a single platform but may be realised with an hybrid approach . The number of devices accessing internet will be 50 billion in 2020 [3]. But only 30% of the total user segment energy is available to operating cost, passive optical networks (PON) allows for high capacity broadband along with considerable energy consumption reduction [15]. The Wireless Sensor Networks are currently being employed in IoT subsystems, as they provide a resilient path to connect the physical analog values to any compute-capable devices anywhere in the real world. The currently deployed WSNs required to work (sense) autonomously and continuously for long period of time. As the space the resources for WSNs are limited, a need for improved energy-efficiency is in demand. Also as WSNs applications vary from small-scale smart home system to large-scale environmental monitoring the energy optimization techniques for WSNs should be application specific. Various techniques such as data-reduction, radio-optimization, sleep/wakeup schemes need to be tried and tested for our cloud based liquid-oil detection system.

Mininet is a virtual network [2] prototyping application which allows simulation and analysis of Software defined networks [6]. It creates a network on virtual resources and provides for data and flow control manageability, it also has the provision for internal and external SDN controllers. One compatible controller with mininet and Openflow is the OpenDayLight controller. Because of its various southbound interfaces such as PCEP, LISP, OF1.0, SNMP [13] and high modularity and orchestration support has made it popular among the vendors. The idea is to perform an experimental low cost demonstration An open source microcontroller which can be employed for the purpose is Arduino board. With its analog to digital conversion capability [11], it is the ideal board for sensing analog values continuously such as temperature, humidity, proximity distance. The communication module which is currently popular in the IoT world is the ESP8266. The ESP8266 Wi-Fi module is a small affordable SoC (system on chip) which essential synchronizes Wi-Fi and serial channels and is able to provide maximum functionality with minimum cost. With all the available hardware the purpose is to integrate the mininet SDN with our IOT subsystem using a cloud platform. The overview of the paper is as follows. Section 2 highlights the scenarios which served a motivation for developing a liquid spill and leakage detection system. Steps to set up an emulated SDN network for our project are explained in section 3. Section 4 presents the actual working of the real time experimental demonstration. Critical analysis of various network topologies and energy saving techniques in WSNs is carried out in section 5. Conclusion and future scope of our project has been discussed in section 6.

2. Motivation

An oil tanker near Suao, east coast of Taiwan witnessed an oil-spill spill in the sea following a fuel leak in December 2006 [18], 256 barrels of heavy fuel oil were spilled into the San Pedro Bay, Los Angeles from a car carrier on May, 2016 which spreaded up to 70 acres and took around 23 days to clean-up [20] ,a Trans mountain pipeline leakage on May 27, 2018 caused the release of 4,800 litres of crude oil [19] and many such incidents have reported in the past and recent times which not only cause resource wastage but also diminishes the environmental well-being. The increase in offshore industrial activity and international trade have led to an increased rise in threats to marine habitats. The oil tanker and container ships have been reportedly found to have affected the aquatic ecosystem as the later were found with leaky drainage and pipeline system which couldn't be detected in the time being the least resulting in large scale threat to ocean-biodiversity has further alarmed the need for an smart automated system to handle such faulty and life –threatening leakages in the near future. Also the recent success of SDN in managing complex networks have sparked the idea of integrating SDN and WSNs.



Fig. 1: Oil tanker vessel spilling oil after a leakage in its container

3. SDN emulator

A network emulator helps to realise the essence of an actual real time network by provisioning resources on a virtual network. Several framework for SDN simulation are available such as CloudSimSDN [8], which simulates cloud environments using the java classes predefined in it. But it suffers from high system workload and requires Java-capable compiler support. Mininet on the other hand, is scalable and provides low-cost virtual networks on limited resources (single PC) and is designed specifically for research purposes in SDN [9]. We will be using the open source mininet network emulator for our project due to its rich features and ability to support openflow protocol.

3.1 Setting up the mininet environment

We installed a mininet virtual machine on the virtual box which is based on Linux server operating system [21]. The mininet VM can be downloaded from mininet website (<http://mininet.org/download>). Two network adapters are required for the mininet network, one connected to NAT (network address translation) and the other as a host only adapter which will serve as a dhcp client which will be assigned an ip address to be able to be recognised in a network for remote communications. Putty software will be used as a terminal to access mininet on this ip address via SSH remotely. There are several key entities in a mininet virtual environment like host, switch, controller, and links. Each host is connected to a layer 2 switch which is a network device capable of establishing links. It is the switch which performs the actual data communication in the network. By default mininet has a built-in controller to manage flow control, but we will be using an external controller for our project which is the OpenDayLight controller.

3.2 Creating network topologies

After setting the mininet environment, the next step is to add hosts ,switches, controller, links to the network in order to reralise a network topology .Various topologies can be created either through the mininet built-in topologies or custom topologies using mininet python API commands [9]. Mininet offers five built-in topology namely single, minimal, reverse, linear, tree. For creating a tree topology with depth of 2 and fanout of 3 we execute the “sudo mn -- topo tree, depth=2, fanout=3”. The connection between any two switches (hosts) could be checked using the “ping” command. Fig. 2 shows tree and linear topology created using “sudo mn -- topo” command

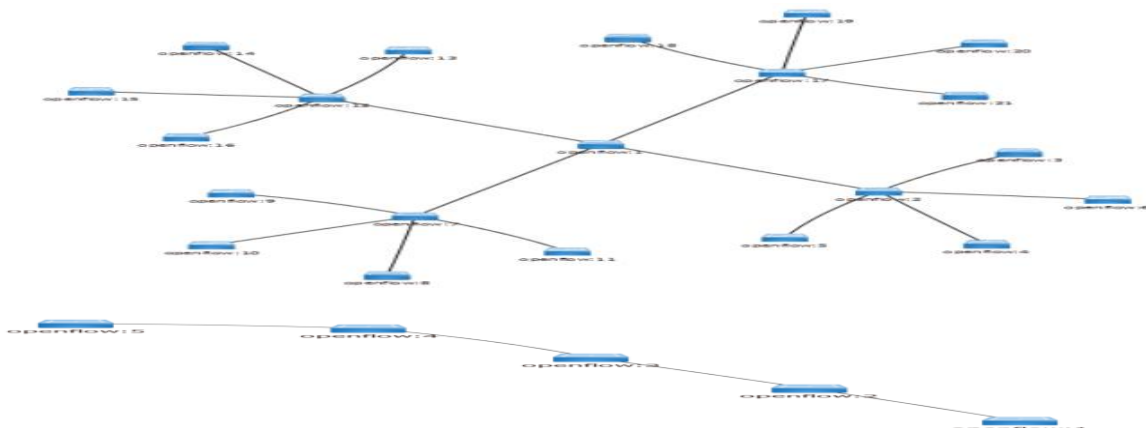


Fig. 2: Topologies in mininet

3.3 Setting up the openflow connection

OpenFlow is a communication protocol that allows control plane to interact with data plane [16]. The forwarding decisions made in the control plane takes effect in data plane via OpenFlow protocol. Mininet nodes support Openflow and allows controller to be either internal or external such as NOX, OdenDayLight as in our case. We installed OpenDayLight controller on a separate Ubuntu desktop virtual machine [22] and set the host only network adapter (eth1) as dhcp client which was assigned the ip address of 192.168.56.103.

Wireshark is a built-in application in mininet which could be used to monitor and analyse data flow between hosts and could be run using the '*sudo wireshark &*' command. In Fig 3. We can see the communication between the mininet (ip address "192.168.56.102") and OpenDayLight controller(ip address "192.168.56.103") being taking place using the openflow protocol (OF 1.3) .

No.	Time	Source	Destination	Protocol	Length	Info
10405	194.60536500	192.168.56.102	192.168.56.100	DHCP	342	DHCP Request - Transaction ID 0x9362
10406	194.97232500	192.168.56.102	192.168.56.1	SSH	2310	Encrypted response packet len=2256
10407	194.97268200	192.168.56.1	192.168.56.102	TCP	60	51160 > ssh [ACK] Seq=1304321 Ack=414
10408	194.97509000	192.168.56.1	192.168.56.102	SSH		OpenFlow between mininet (192.168.56.102) and OpenDayLight (192.168.56.103) 128
10409	194.97514200	192.168.56.102	192.168.56.1	TCP		Ack=130
10410	194.99799400	192.168.56.103	192.168.56.102	OF 1.3	122	of_flow_stats_request
10411	194.99859400	192.168.56.102	192.168.56.103	OF 1.3	418	of_flow_stats_reply
10412	194.99929900	192.168.56.103	192.168.56.102	TCP	66	6633 > 49890 [ACK] Seq=1401 Ack=8951
10413	195.01083600	192.168.56.103	192.168.56.102	OF 1.3	90	of_port_stats_request
10414	195.01113900	192.168.56.102	192.168.56.103	OF 1.3	418	of_port_stats_reply
10415	195.01457400	192.168.56.103	192.168.56.102	OF 1.3	90	of_queue_stats_request

Fig. 3: Wireshark analysis of Openflow

The OpenDayLight controller provides a GUI for users which can be accessed by browsing the "192.168.56.103:8181/index.html" url on any browser. After logging in as admin, various options to manage flow control will be made available such as topology view, packet forwarding, yang UI, network visualizer, node connector.

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:2:1	12	157	840	12805	0	0	0	0	0	0	0	0
openflow:2:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0
openflow:2:3	144	145	11880	11965	0	0	0	0	0	0	0	0
openflow:2:2	133	157	11125	12805	0	0	0	0	0	0	0	0

Fig. 4: Node connector statistics for node Id – openflow:2

3.4 Connecting to the thingspeak cloud

Each host in a mininet is able to connect to internet using NAT adapter and run python scripts. Every host is made to parse a Json file from the thingspeak cloud using the “`json.loads()`” and “`urllib.urlopen()`” function in a “`client.py`” python file. The Json file consists of an array which stores the latest values of the sensor fields as uploaded by real-time sensors using the ESP8266 which will be explained in section 4.2. These sensor values parsed from the Json file are then forwarded to a centralised server.

ThingSpeak (<https://thingspeak.com>) is a free public cloud offering storing and accessing facilities to IoT sensor data. It provides an Open API to collect real-time sensory data along with their analysis and visualization with Matlab [17]. For our project we created a channel which stores three sensor fields namely distance (distance between container roof and liquid stored in it), humidity and temperature. As the sensor continuously sense the environment and upload it to the thingspeak channel, we get various sensor values depending upon the time the surrounding was sensed.

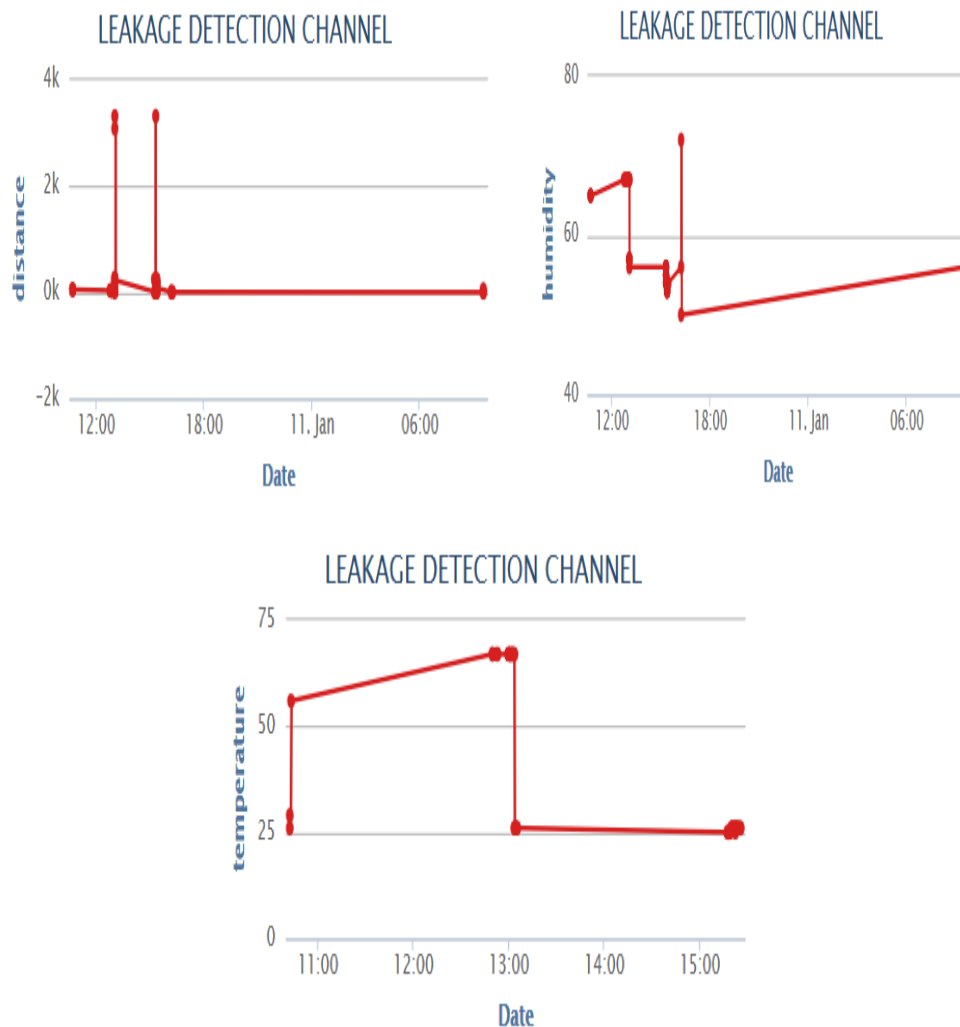


Fig. 5: Data updation on the thingspeak cloud

3.5 Working the client-server architecture

A host in a mininet network is assigned task of a server whose job is to continuously receive sensor values from clients which are connected to the thingspeak cloud as mentioned previous section and generate an alarm if any hazardous value of either of the sensors is received. For example if temperature rises to 80 degree Celsius , then an alert message is generated on the server terminal which will alarm the concerned authorities and immediate follow up action could be taken without any much loss of time.

The clients and server may me organised in any of topologies as discussed in section 3.2. The aim is to reduce network traffic and hence jitter. Analysis of various topologies have been done in section 5 and star topology came out to be suitable of our project. We created a custom star topology using mininet python API commands [9] as it is not a mininet built-in topology. Then we executed “server.py” python script on host h1 with ip address “10.0.0.1” .We then executed the “client.py” python script using the “sudo python client.py command” on each client hosts which essentially reads sensor values from thingspeak cloud and sends it to server on host with ip addresss “10.0.0.1” as shown in Fig 6.

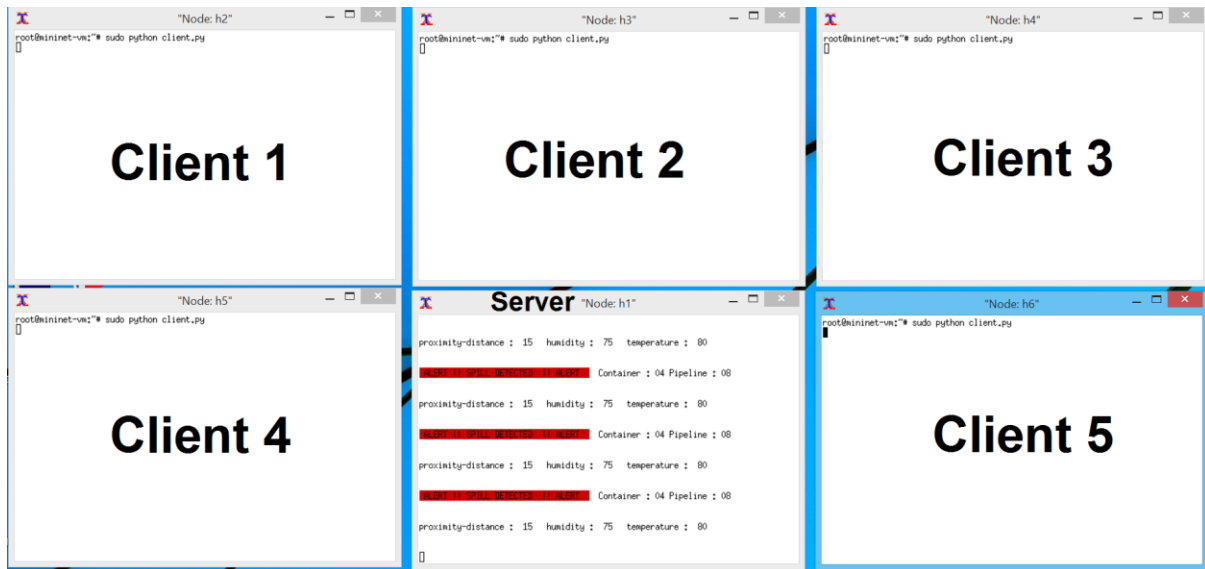


Fig. 6: Siix clients connected to a server sending sensor data continously

4. Experimental setup

For the experimental demonstration we set up a simple sensor mote consisting of all the essential requirements from sensing analog value to writing to the cloud. The hardware requirements of the project are: Arduino uno microcontroller, ESP8266, HC-SR04, DHT-11, LED light, jumper wires (M2M, M2F), resistors (10k Ohms and 2k Ohms).

The Arduino microcontroller operates on 5V power source and has a clock speed of 16 MHz [11], also it supports IEEE 802.15.4.433RF BLE 4.0 and other wifi communication technologies. The power/usb cable may either be connected to a power source or to a usb port to upload code on to the microcontroller. The Wi-Fi module employed in the project is

ESP8266. It comes with two UART pins Rx and Tx, two GPO pins, PCB antenna and a 32-bit based CPU [10]. The Rx and Tx pins can be connected to any digital pin on the Arduino board, it operates on 3.3V (unlike DHT-11 and HC-SR04 which operates on 5V). ESP8266 may be powered in various modes such as standby, sleep, deep sleep, transmit, receive, as the situation demands. Additionally a 10K Ohms resistor might be used to keep the Rx line high. The ultrasonic range finder used in our project is HC-SR04 which generates ultrasonic waves of frequency 40 kHz using a transmitter and employs a receiver to perceive the generated echo [5]. The sensor reports the time taken for the sound waves to travel to the object and back. The Trig and Echo pins can be connected to any digital pin on the Arduino board. DHT11 is widely used to get an digitalized and calibrated output of analog measures of temperature (in degree Celsius) and humidity (relative) [7] using the built-in humidity sensing component and thermistor. It consists of three pins, the power and ground pins are connected to 5V and ground pins of the Arduino board, the data pin maybe conned to any one of the digital pins.

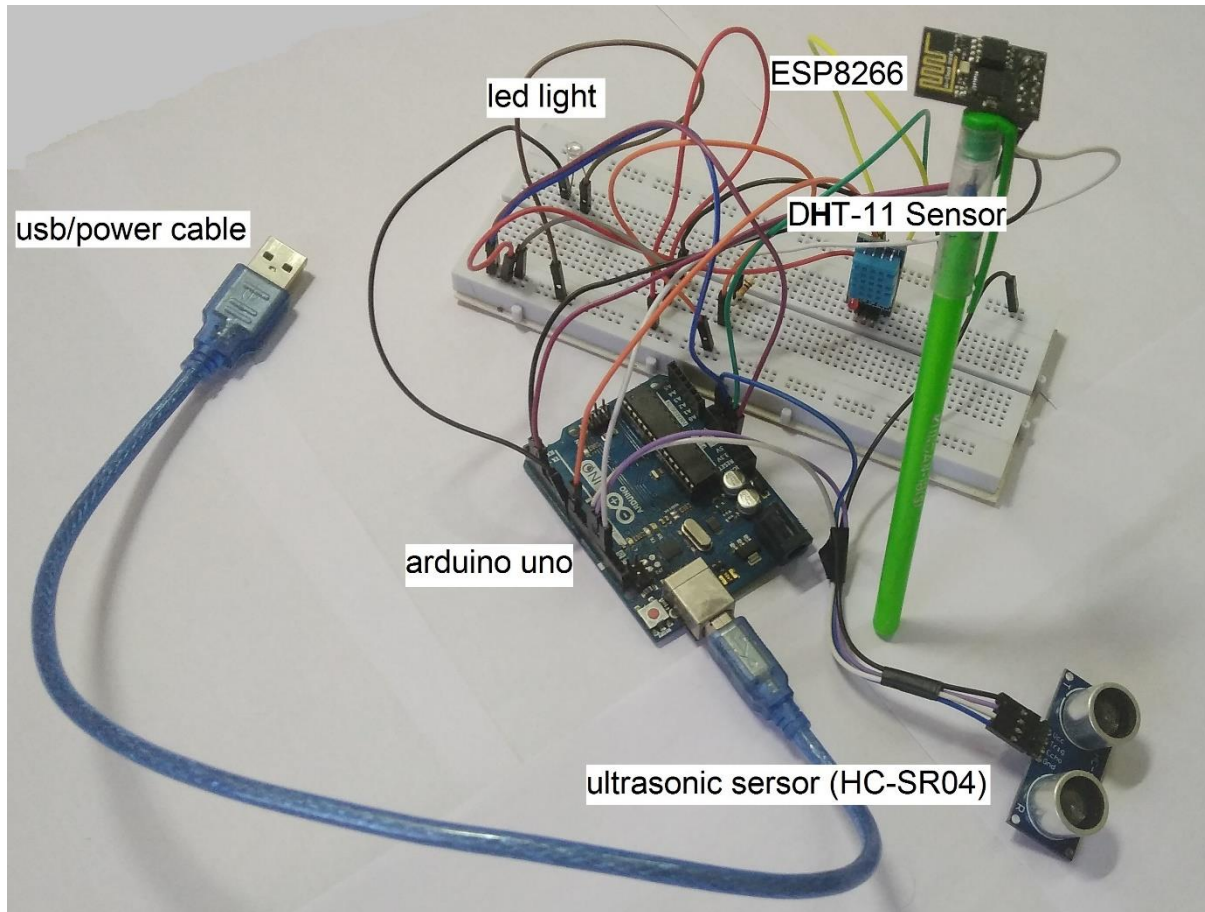


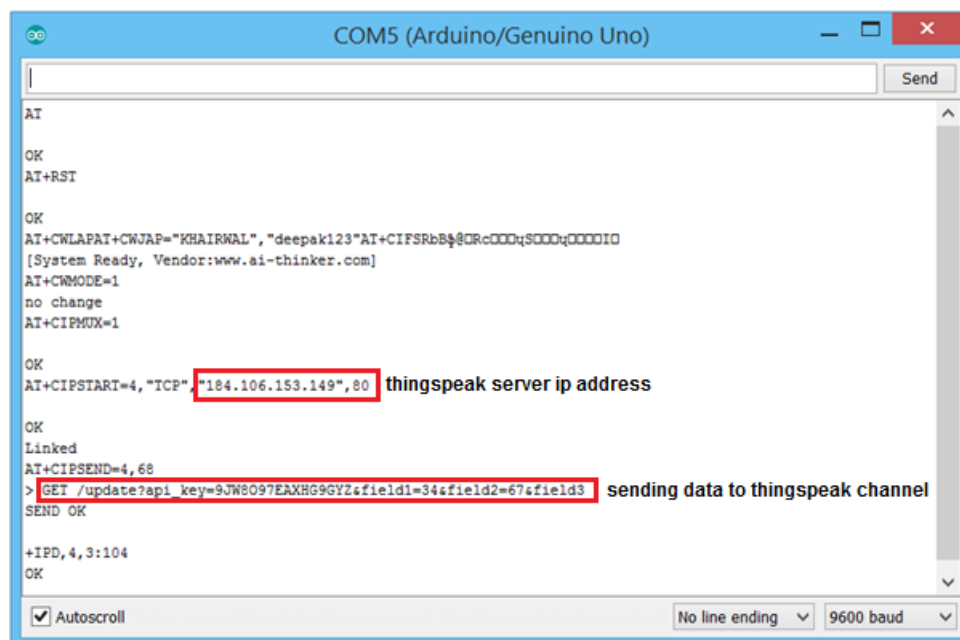
Fig. 7: Hardware layout of experimental setup

4.1 Working with Arduino IDE

Arduino uno comes with integrated development environment (IDE) with built-in C/C++ compiler which allows code to be compiled and uploaded onto the Arduino board. The Arduino IDE also has the provision of a serial monitor which is used to monitor the serial communication between the board and its modules. The ESP8266 module is controlled and monitored by a set of AT commands which directs the behaviour of the Wi-Fi communication. The DHT sensor is managed by a dht library which has to be included in the program code. For our project ESP8266 works in two modes : AP (access point) and STA (station) . It may be programmed to operate in any mode depending upon the application it serves,

4.2 Remotely located sensor motes

In a real world application, there will be several of the sensor motes operating in different locations, these motes need to continuously sense the environment and send the sensor data to the centralized authority which will detect any alarming values of the sensors. In the SDN network created in section 3, every single host will be connected to a sensor mote through the thingspeak cloud. The thingspeak cloud provides the provision of channels in which eight sensor values (called fields) of any type can be read or written using the private API keys. For writing into the cloud ESP8266 operates in station mode (STA) to write data into the sensor fields of the channel using the channel write API key. For the reading the purpose a Json file will be fetched from the thingspeak server using the read API key by the computable hosts in the SDN network, which would be sending the data continuously to the centralized server. Fig. 8 shows how the Wi-Fi module connects to the thingspeak server and writes the sensor data into the channel, which are then subsequently fetched by parsing the Json file received from the thingspeak cloud by the client hosts connected to a server in a Software defined Network as mentioned in section 3.4.



```
COM5 (Arduino/Genuino Uno)

AT
OK
AT+RST

OK
AT+CWLAPAT+CWJAP="KHAIRNAL", "deepak123"AT+CIFSRbB$@CRc000q$000q000010
[System Ready, Vendor:www.ai-thinker.com]
AT+CMODE=1
no change
AT+CIPMUX=1

OK
AT+CIPSTART=4, "TCP", "184.106.153.149", 80 thingspeak server ip address

OK
Linked
AT+CIPSEND=4,68
> GET /update?api_key=9JW8097EAXHG9GYZ&field1=34&field2=67&field3 sending data to thingspeak channel
SEND OK

+IPD,4,3:104
OK

Autoscroll No line ending 9600 baud
```

Fig. 8: Serial monitor showing ESP8266 sending data to thingspeak cloud in STA mode

The ultrasonic sensor will be set on the top roof of the container, if the liquid level of the container goes down below a threshold value then leakage alert will be initiated at the centralized server, similarly if the humidity or temperature (in case of fire) exceeds alarmingly, the server will be notified of the sensor location and the conditions of the environment as shown in Fig. 6. Algorithm 1 highlights the procedure for SDN network to be able to detect any hazardous values of proximity distance, humidity or temperature.

```

- Input : Sensor values for proximity distance, humidity and temperature
- Output : Response to the sensor values from clients
for i=1 to num( clients ) do
    data  $\leftarrow$  thingspeak cloud data for sensor mote i
    send (data)
end for
for j=1 to num( clients ) do
    receive( data )
    if data.value > threshold then
        alert( "leakage detected")
    else
        print(data)
    end if
end for

```

Algorithm 1: SDN based client-server leakage detection

4.3 Devices within LAN

As for the case of a closed environment like a merchant ship, small factory, the sensor motes can directly provide the condition of their surrounding without the need of a cloud platform.

By operating ESP8266 in access point mode (AP), a Wi-Fi hotspot name "ESP8266" will be created which will be available to all service personnel or employees who then can directly get the container or pipeline status on their smartphones, laptops, or tablets.

Fig 9 shows how ESP8266 creates an access point for the devices within the LAN to be able to connect to the router ip address: "192.168.4.1".



Fig. 9: Serial monitor showing ESP8266 setting hotspot for LAN devices in AP mode

Devices within 300 meters of the access point can search the Wi-Fi hotspot and can directly access the online spill detection interface by connecting to the “192.168.4.1” ip address.

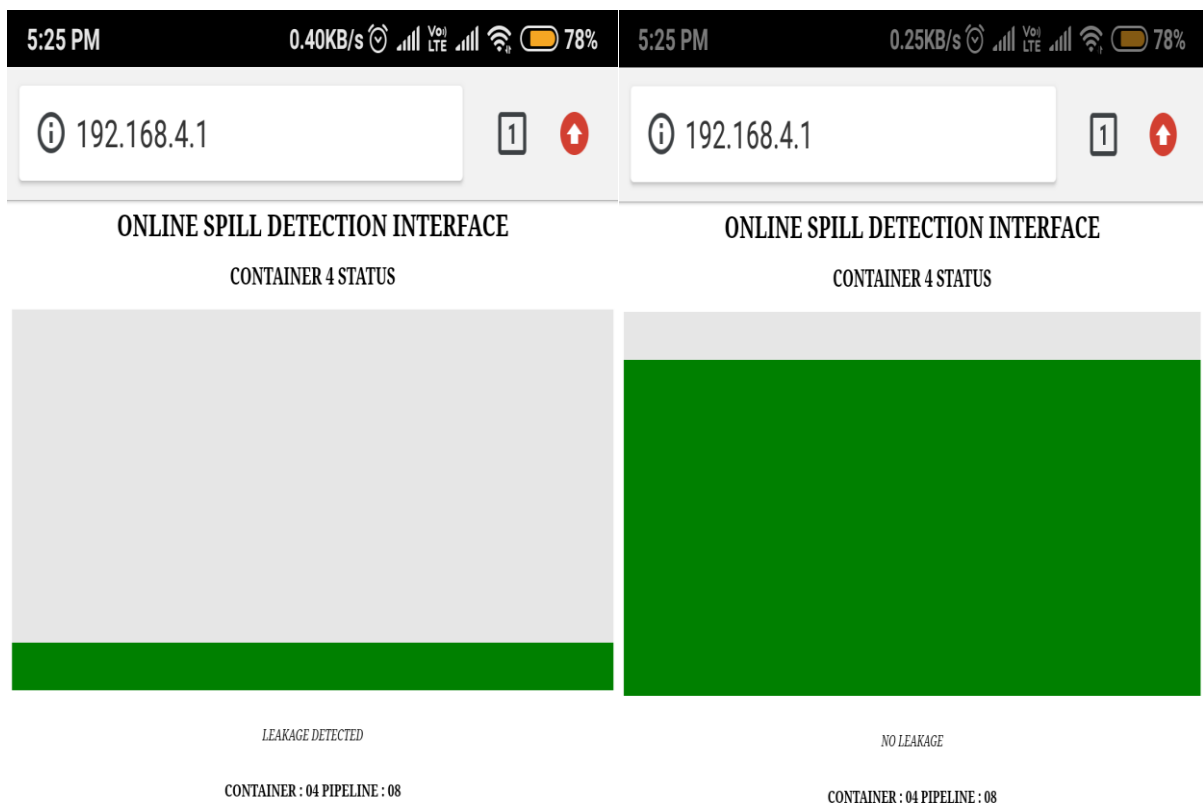


Fig. 10: Screenshot of LAN device connected to ESP8266

Fig. 11. Depicts two scenarios wherein , one, wherein the users are within the access range of the Wi-Fi LAN and hence of the sensor motes can directly access the spill detection interface by connection to the ESP8266 Wi-Fi hotspot achieved by operating ESP8266 in AP mode. The other scenario is of the remotely located sensor motes which cannot be connected to user end devices using LAN and hence require cloud platform to be able to get accessed by multiple host continuously reporting sensor data to the server organised in a SDN network whose flow control is managed by an external SDN controller (OpenDayLight as in our case).

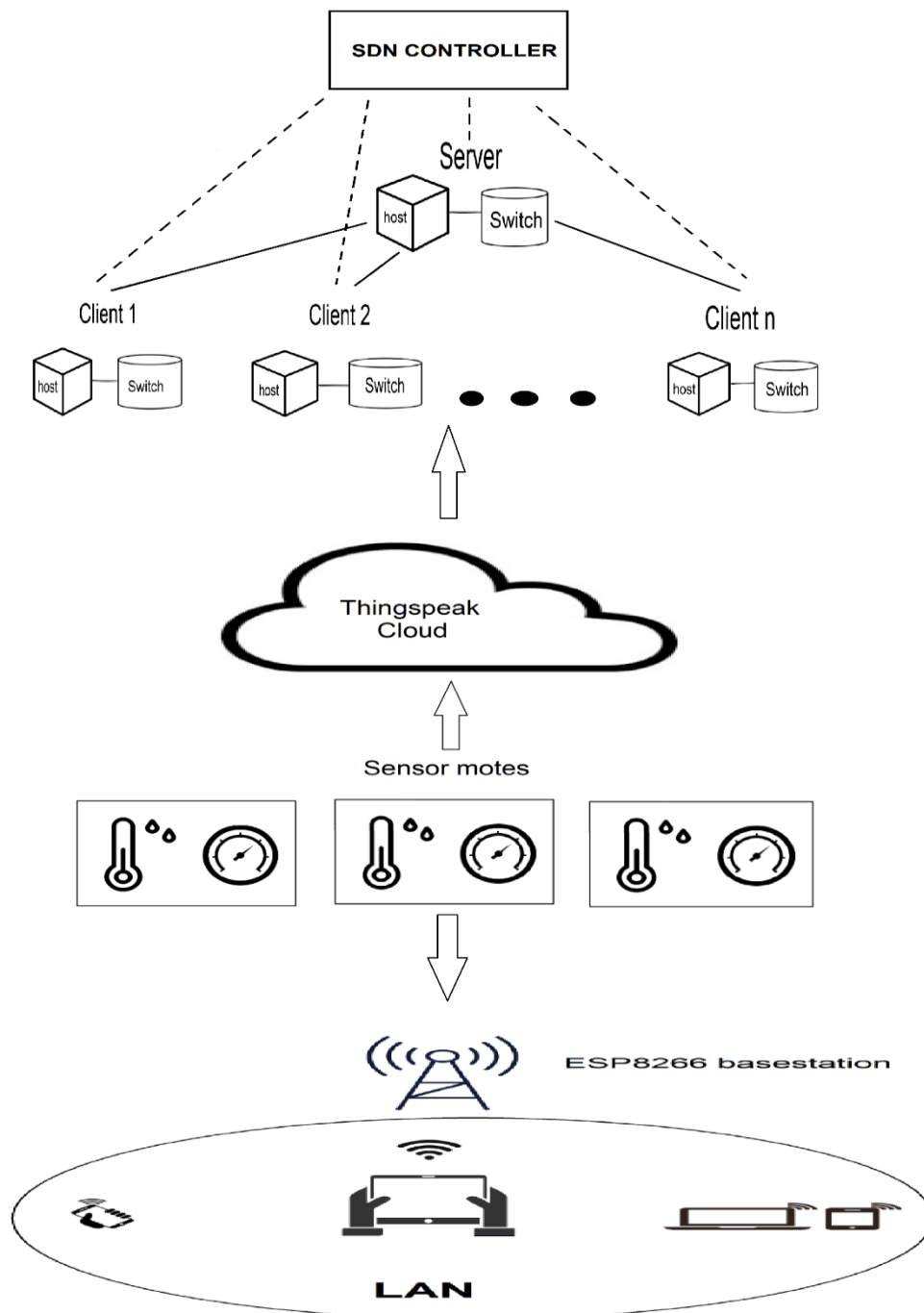


Fig. 11: Architecture overview

5. Result and analysis

Network topologies plays an important role in deciding the latency and hence the effective communication time. According to the network topology, the IoT application provisions the optimal path [4] in the SDN virtual network, configures the Openflow switches and uses the Openflow protocol to set rules in the respective flow tables [1].Jitter is seen as an important QoS when it comes to networking. For our SDN network to operate flawlessly, the effort should be to reduce the jitter to the minimal. The iperf commands provide an effective way to measure the performance of communication between two hosts.

We created four network topologies namely linear, star, tree and ring using the mininet command line .Then we set one host in each topology as server and executed the “iperf -s -u -i 1” command in it, which essentially is a command for a server to listen on UDP port with interval of one second between periodic bandwidth, jitter, and loss reports. By default the first host (h1) in mininet is given the ip address of “10.0.0.1”. Then we chose a host on each topology and executed the “iperf -c 10.0.0.1 -u -b 1m -n 1000000” , which creates a udp client that connects to h1 at address 10.0.0.1 with bandwidth of one megabytes and instructs it to transfer 1,000,000 buffers. The result we obtained is shown in Fig 12.

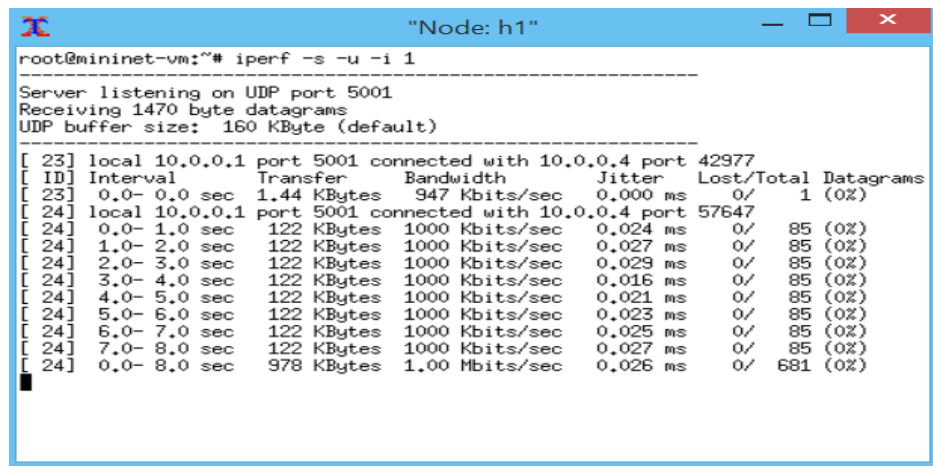


Fig. 12: A server host transmission report

Table 1 shows the performance of various topologies. Clearly star topology has witnessed a considerably low jitter, as expected. Since there exists a one-one connection between each host to the server, no matter where the host is located in the network, a uniform and shortest communication path to the server is guaranteed, unlike in other topologies where the data packet has to be forwarded to the intermediate hosts in order to reach the server and this consumes unnecessary bandwidth and increases the network traffic and hence the jitter. Although star topology suffers from single point of failure and increased cost due to comparatively more node connections, a trade-off can always be considered.

Topology	Bandwidth	Propagation time	Jitter
linear	1 Mb	1 sec	0.026 ms
star	1 Mb	1 sec	0.024 ms
tree	1 Mb	1 sec	0.029 ms
ring	1 Mb	1 sec	0.028 ms

Table 1: Analysis of various topologies

As far as the energy savings of the sensors and actuators is concerned, several techniques such as radio optimisation, data reduction [12], energy-efficient routing and sleep-wake up schemes are in use. The preferred among them which could be used in our project are

Energy harvesting: allowing sensors to generate energy from renewable resources such as solar, kinetic, and thermal.

Sink mobility: employing a moving base station to collect essential node information, proving load-balancing and hence improve network lifetime.

Data compression: encoding the data to be send by using various schemes such as Manchester, Huffman coding to reduce the bytes to be sent and hence reduce the power requirement.

Modulation optimization: it involves searching the parameters that would result in minimal power consumption of the radio subsystem

Network coding: In broadcast scenarios where copy of every packet is sent, forwarding a linear combination of various packets can be used as a traffic reduction scheme and hence is energy efficient.

Sleep-wake up schemes: in some scenarios, the sensors need to send data occasionally rather than continuously, these idle states consume unnecessary energy. The ESP8266 could be put on standby, deep sleep or power save mode DTIM 1,3 [10] to considerably reduce energy consumption.

An important aspect which is considered while sandwiching various technologies is throughput. In order for a system to be scalable it must overcome throughput and jitter degradation. SDN architecture combined with multipath TCP [24] can significantly improve throughput as it has the overall knowledge of the underlying network topology. The controller automatically finds the shortest path by creating OpenFlow rules and group tables. The on spot available capability of each network link serves as a search criteria for placing auxiliary subflows belonging to the same connection.

With only IoT and cloud as the leading weightlifters the system suffers from manageability issues and hence computation failures , with SDN in the picture the jitter is reduced as the control plane which accounts for routing table and shortest path is isolated from the system and is operated remotely which efficiently takes care of various scalability issues. For providing load sharing in cloud environment a load sharing algorithm [23] is employed to reduce delay and jitter which utilizes the capability of a SDN controller which finds an optimal path whenever an unmatched packet is received at a switch, and the entry is made into the open flow table for future references.

6. Conclusion and future work

Through this paper, we demonstrated how various technologies could be integrated to provide effective solutions to real world social and ecological hazards. Although mininet provides an option to emulate SDN experiments on virtual network, and also provides topological manageability, it lacks controllability for a large scale system. Implementation of the virtual network on real time hardware of a large scale system is a challenge. The popularity of OpenDayLight is due to grow not just because of its rich features but also due to its vendor supported interfaces. With the rapid emergence of IoT and WSN, ESP8266 would be seen in excessive in demand in the field of research and development as it provides maximum functionality in minimal cost. However it can be integrated with external antennas to maximize the range of the LAN by amplifying the carrier signal waves.

The future idea could be to integrate automated spill and leakage detection system with oil skimming application managed by swarm robots [14] to make the whole system autonomous. Deploying the demonstration in real world scenarios and going through stages of testing and enhancement. Exploring new energy-saving schemes for WSNs and utilizing the cloud platform for storing more sensor values.

References

- [1] Ogrodowczyk, Ł., Belter, B., & Leclerc, M. (2017). Iot ecosystem over programmable SDN infrastructure for Smart City applications, 49–51.
<https://doi.org/10.1109/EWSDN.2016.17>
- [2] Alenezi, M., Almustafa, K., & Amjad, K. (2018). Cloud based SDN and NFV architectures for IoT infrastructure. *Egyptian Informatics Journal*.
<https://doi.org/10.1016/j.eij.2018.03.004>
- [3] Tomovic, S., Yoshigoe, K., & Maljevic, I. (2016). Software-Defined Fog Network Architecture for IoT. *Wireless Personal Communications*.
<https://doi.org/10.1007/s11277-016-3845-0>
- [4] Bi, Y., Han, G., Lin, C., Deng, Q., Guo, L., & Li, F. (2018). Mobility Support for Fog Computing : An SDN Approach, (May), 53–59.
- [5] Series, C. (2018). Application of ultrasonic sensor for measuring distances in robotics.
- [6] Zulu, L., & Umenne, P. (2018). Simulating Software Defined Networking Using Mininet to Optimize Host Communication in a Realistic Programmable Network ., (August). <https://doi.org/10.1109/ICABCD.2018.8465433>

- [7] Sharma, A., Tiwari, G., & Singh, D. (2016). Low cost Solution for Temperature and Humidity monitoring and control System using Touch Screen Technology, 2(1), 10–14.
- [8] Jemal, A. A., & Morshed, A. (2018). SDN enabled BDSP in public cloud for resource optimization. *Wireless Networks*, 9. <https://doi.org/10.1007/s11276-018-1887-9>
- [9] Kaur, K., Singh, J., & Ghumman, N. S. (n.d.). Mininet as Software Defined Networking Testing Platform, 3–6.
- [10] Mehta, M. (2015). ESP 8266 : A BREAKTHROUGH IN WIRELESS SENSOR NETWORKS AND, 6(8), 7–11.
- [11] No, I. (2017). Available Online at www.ijarcs.info International Journal of Advanced Research in Computer Science A Comparative Study of Arduino , Raspberry Pi and ESP8266 as IoT Development, 8(5), 2350–2352.
- [12] Rault, T., Bouabdallah, A., & Challal, Y. (2014). Energy-Efficiency in Wireless Sensor Networks : a top-down review approach. *COMPUTER NETWORKS*, (March). <https://doi.org/10.1016/j.comnet.2014.03.027>
- [13] Mamushiane, L., Lysko, A., & Dlamini, S. (2018). A Comparative Evaluation of the Performance of Popular SDN Controllers, 54–59.
- [14] R, P. P., Sanjay, A. R., Gupta, Y. V., & Rekha, D. (2018). Investigating feasible tool for swarm robotic based oil skimming application, 7, 960–967.
- [15] Dourado, D. M., & Rondina, U. (2017). Energy consumption and bandwidth allocation in passive optical networks. *Optical Switching and Networking*, 28(October), 1–7. <https://doi.org/10.1016/j.osn.2017.10.004>
- [16] Aydeger, A., Akkaya, K., & Uluagac, A. S. (2015). SDN-based Resilience for Smart Grid Communications, 31–33.
- [17] Ray, P. P. (2017). ScienceDirect A survey of IoT cloud platforms. *Future Computing and Informatics Journal*, 1(1–2), 35–46. <https://doi.org/10.1016/j.fcij.2017.02.001>
- [18] <https://inews.co.uk/news/long-reads/cargo-container-shipping-carbon-pollution>
- [19] <https://www.northislandgazette.com/news/trans-mountain-pipeline-spill-much-larger-than-b-c-government-first-reported>
- [20] <https://worldmaritimenews.com/archives/251165/los-angeles-files-lawsuit-over-2016-car-carrier-oil-spill/>
- [21] <http://www.brianlinkletter.com/set-up-mininet>
- [22] <https://www.brianlinkletter.com/using-the-opendaylight-sdn-controller-with-the-mininet-network-emulator>

- [23] Amiri, M., Osman, H. Al, Shirmohammadi, S., & Abdallah, M. (2015). An SDN Controller for Delay and Jitter Reduction in Cloud Gaming An SDN Controller for Delay and Jitter Reduction in Cloud Gaming, (October).
<https://doi.org/10.1145/2733373.2806397>
- [24] Alharbi, F. (2018). An SDN Architecture for Improving Throughput of Large Flows Using Multipath TCP. *2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 111–116.
<https://doi.org/10.1109/CSCloud/EdgeCom.2018.00028>
- [25] <https://www.uniindia.com/another-pipeline-blast-caused-by-illegal-tap-rocks-mexico-s-hidalgo-state/world/news/1482110.html>