Securely accessing remote sensors in critical infrastructures.

Research project for the SNE masters programme

Pavlos Lontorfos
Supervisors: Cedric Both and Jeroen de Boer

July 5, 2020

Sensor technology has become an invaluable tool for monitoring an environment, enabling users to identify inefficiencies within their systems and to reduce risks related to operation and maintenance. These sensors are widely used in public infrastructures such as bridges, tunnels, and electricity networks. This expansion of their use in critical infrastructures has increased the need for secure and scalable communication with those sensors. In this paper, I will perform theoretical research as well as a series of experiments and I will build a software defined network(SDN) infrastructure using LoRa sensors in order to examine the effects of software defined networking on sensor networks. I will focus on the aspects of redundancy, scalability, and security of those networks and I will evaluate the benefits that an SDN-based architecture can have in such architecture.

Contents

1 Introduction

Critical infrastructures, such as water supply, transportation, and electricity generation networks are essential for the functioning of a society and economy and therefore are expected to be highly available in every situation. Hence, they need to be protected not only from accidental failures, such as disaster but also from malicious actors[1]. Consequently, monitoring mechanisms that enable network operators to detect failures and attacks as early as possible must be in place. Wireless sensors are already widely in use, but the emergence of cloud computing, highly intelligent 'Smart' devices, and 5G networks is expected to increase demand for situational awareness leading proliferation of sensors, especially in the critical components of the society[2].
Although the use of wireless sensors could solve the problem of extended monitoring of the critical infrastructure, it creates two new challenges. The first is that often the infrastructure that needs to be monitored is inaccessible, like tunnels or underwater constructions, and a failure on the sensor network can lead to complete loss of visibility of the asset. Understandably, redundancy of the communication is of major importance.
The second challenge is that sensitive information about critical parts of a country's infrastructure is transmitted over the internet. This information can attract malicious actors who would like to get access to it. Due to the aforementioned scenario, a secure way of communication, as well as continuous monitoring of this communication must be in place.

1.1 Research Question

In this research, I will evaluate the benefits that the use of SDN can have in a remote sensor network. The main research question will be:
Can the use of Software Defined Networking improve the redundancy and security of a sensor network used in critical infrastructure objects?
To answer the main question, I first need to answer the following sub-questions:

1. What benefits can an SDN-based architecture have regarding the redundancy of the system?

2. What benefits can an SDN-based architecture have regarding the scalability of the system?

3. What benefits can an SDN-based architecture have regarding the security of the system?

2 Background

In this section I will provide a theoretical background of the protocols and technologies that will be used during this research, as well as relevant work that has been done by other researchers.

2.1 Relevant knowledge

Software-Defined Networks
Software-Defined-Networks were introduced to give the network operators greater flexibility and agility in the services they provide. To achieve this, SDNs introduce new characteristics to traditional networks. The first key feature is the separation of the control plane from the underlying network data plane for efficient

data transport and better control of network management and services[3][4]. Building on this idea, the next characteristic is the simplification of devices, which are controlled from a centralized system running management and control software. The complex logic that allows every device to behave autonomously is moved from the switches to a centralized controller which manages the network and provides basic instructions to the simplified devices on how to deal with the incoming traffic[5]. An SDN has two distinct interfaces, the northbound interface which describes the area of protocol-supported communication between the controller and applications or higher layer control programs mostly through APIs, and the southbound interface which defines the communication protocol between forwarding devices and control plane elements[6]. A simplified representation can be found in figure 1.

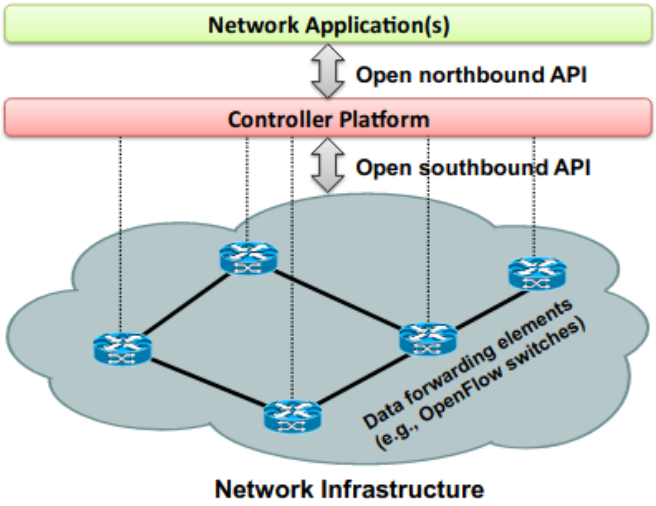


Figure 1: Simplified view of an SDN architecture [6].

OpenFlow
Openflow is a framework through which a logically centralized controller can communicate with and control an OpenFlow switch. It originally defined the communication protocol in SDNs that enables the network controller to directly interact with the forwarding plane of network devices such as switches and routers, both physical and virtual. Each OpenFlow capable device can maintain flow tables, which are used to perform packet lookups[7].
Network Functions Virtualization
Network functions virtualization (NFV) is a way to virtualize an entire class of network components, such as routers, firewalls, and load balancers, that have traditionally been run on proprietary hardware. It allows the network operators to create new functions as software instances in a virtual machine(VM), and deploy them in their infrastructure, or a data center, without the need for specialized hardware[8]. As a result, the operators can easily deploy additional virtual machines(VMs) to serve the demand, move them at will, or terminate them when the function is no longer needed [4]. NFV supports SDN by providing

the infrastructure on which SDN can run.

Wireless sensor network architecture

A wireless sensor network (WSN) typically consists of a large number of low-cost, low-power, and multi-functional sensor nodes that are deployed in a region of interest. These sensors communicate over a short distance, usually over a wireless medium and they collectively gather information about a specific task, such as environment monitoring, industrial process control, home intelligence, and more [9]. A network like this has some important design characteristics. The most important characteristics for our case are the following:

Self-Configurability: The nodes can be placed in a region, without excessive planning. After deployment, a node can join, leave, or fail at any time. The rest of the sensors should be able to adapt to topology changes and node failures.

Scalability: In a sensor network, the number of sensors can vary widely. Even in the same project, the number of nodes can increase significantly over time. The network design should be able to handle different network sizes and scale accordingly.

Quality of Service(QoS): In a big sensor network, it is possible to produce a lot of traffic. Some messages though could be more important than others in terms of delivery latency and packet loss. For example, a fire alarm message must have a higher priority than a humidity sensor in a congested network. Thus, network design needs to take QoS into consideration.

Security: In many applications, sensor nodes can collect and transmit sensitive information. A sensor network should introduce effective security mechanisms to protect the data information in the network or a sensor node from unauthorized access or malicious attacks[10].

Wireless Mesh Networks

One of the current technologies used for wireless communication with sensors is the Wireless Mesh Networks (WMNs) which are communication networks that comprise radio nodes, arranged in a mesh topology. Each node is connected to every other node in range, usually over the IEEE 802.11 protocol. Although, this WMN can create redundant communication to nodes that connect the sensor network to the rest of the infrastructure, some important issues arise. To synchronize the communication each node creates additional traffic. Thus, in a big network, the nodes can exhaust the bandwidth and degrade the communication. Additionally, careful placement of the gateways is important, as it can lead to resource starvation of some nodes while others are barely used[11]. In 2013, Detti Andrea et al. published research with the benefits of an SDN-based implementation of a WMN. They prove that the use of a centralized network controller that can create arbitrary paths for data flows can lead to the development of improved traffic engineering algorithms in WMNs[12]. In 2014, they further improve their research in partitioning and merging scenarios. In detail, they use the ad hoc routing protocol OLSR to establish basic IP connectivity between nodes to forward the control plane, as shown in figure 2. Moreover, the OpenFlow switches in the Wireless Mesh Routers(WMR) interact with the OpenFlow controllers that can configure the flow table for specific data plane flows[13]. Their research proves that a software-defined network can optimize resource management and reduce the convergence time on WMNs.

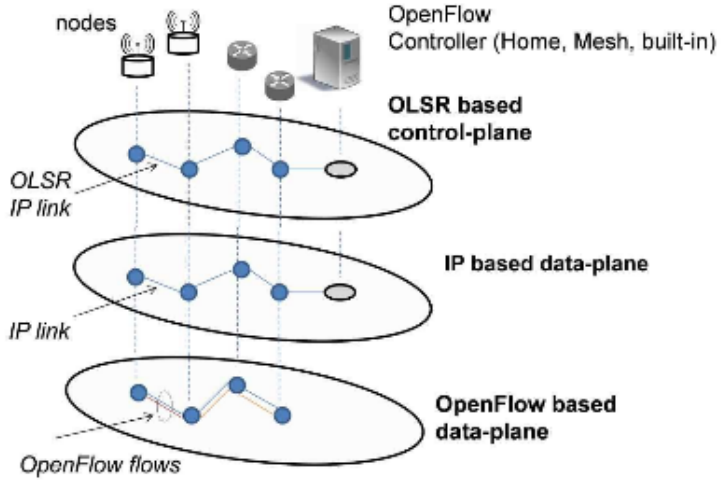


Figure 2: Control and Data plane in wmSDN [13].

2.2 Related Research

In 2015, Izzat Alsmadi et al. created a research paper on the security characteristics of Software Defined Networks[14]. In their research, they discuss the security threats to SDNs according to their effects. I will evaluate if these threats are still relevant to this case and I will analyze how we can mitigate the risks.
In 2017, Mohamed Labraoui et al. made a research about the integration of Wireless Mesh Networks in an SDN-based solution. In their research, they created a self-configurable network where each node could join the wireless mesh, and find its way to the rest of the network using key nodes, which were controlled over SDN [15]
In 2017, Zhiwei Zhang et al. developed a paper that proposes an Efficient Software-Defined Wireless Sensor Network (ESD-WSN) architecture for IoT applications [16]. In this architecture, their goal is to establish a stable and energy-efficient control plane to reduce the control overhead. I will examine their architecture and I will use any beneficial for our case components.
This paper contributes to the previous research as it differentiates in the following parts: I will examine different technologies and I will make conclusions about the benefits that SDN can have in sensor networks used in critical infrastructures. In addition, I will execute the experiments in actual hardware instead of a fully virtualized environment, and I will focus on LoRa sensors, using typical sensors that can be found in critical infrastructures, such as temperature and air quality sensors. According to my knowledge, at the time of writing, there is no similar research publicly available.

3 Methodology

For the purpose of this research I will focus on the effects on security and redundancy from the use of a SDN-based topology on a sensor network. I consider the well-known software and protocols used as secure. Continuous research on the OpenFlow protocol [17][18] shows that its security is constantly

evaluated, and security flaws are fixed.
Another important factor that has to be taken into consideration is that these sensors are usually deployed in remote and inaccessible places, such as tunnels or sewers. According to Datadigest, a major dutch provider of monitoring services in such critical infrastructures, the average time for an on-site visit in case of a failure, is two weeks. During this time, the monitoring of the infrastructure could be limited or nonexistent, depending on the failure. Understandably, the communication with the sensors should be redundant to avoid such catastrophic scenarios.
Due to the wide variety of possible technologies that can be used to communicate with wireless sensors, such as Wi-Fi, Narrowband-IoT(NB-IoT), Lora[19] and more, in combination with the limited time frame of this project, I rely on relative research to cover a few scenarios, while I will focus my experiments on a network that uses LoRa technology to communicate with the wireless sensors.
Hardware and Software Used
For the experiments, in order to create easily reproducible results, I used only open source software and consumer grade hardware. In more detail I used:
Two Raspberry Pi 4, with Dragino Lora/GPS HAT. The Pi's will work both as OpenFlow capable switches and as LoRa gateways. Additionally, I will use a Dragino LoRa gateway with OpenWRT firmware version 19.07, which also supports OpenFlow protocol and OpenVSwitch(OVS). I will also use five LoRa sensors that gather environmental data and forward them to the Lora gateway. I chose LoRa sensors that are typically used in critical infrastructures such as humidity, temperature, and motion sensors as well as an industrial purpose LoRa I/O controller. More specifically, I used the following sensors:

- 2 x Tabs TBHH100-868 (Temperatuire and Humidity sensor)
- Tabs TBMS100-868 (Motion sensor)
- Tabs TBHV100-868 (Air quality sensor)
- Dragino LT-22222-L (LoRa IO controller)

In addition to the aforementioned hardware I used the following open source software:

- OpenWRT firmware[20]
- Open vSwitch[21]
- Faucet SDN controller[22]

On top of our essential infrastructure setup, I will use some additional open source tools that will help me monitor the network easily, named Gauge and Grafana. Gauge is an open source monitoring controller which creates an OpenFlow connection to the switch that monitors ports and flow states, and exports the results to a database or text log files. Grafana is a visualization and analytics software which allows to query, visualize, alert on, and analyze our metrics.

4 Experiments

In this section, there is a detailed description of all the experiments that we executed during the research. These experiments helped me enrich my knowledge about the benefits or drawbacks that SDNs can have in a sensor network

infrastructure. The first experiment focus on the control over the network while the second experiment focus on the redundancy of the communications.

4.1 Network control experiment

The *Network control experiment* allowed to evaluate the benefits that SDNs can have in visibility and control over the the sensor network, as well as its behaviour in unexpected events such as loss of the SDN controller. For this experiment, I will create a SDN which consists of an OpenVSwitch, an OpenFlow controller, three NFV instances, two LoRa gateways, and some sensors. I will use one Raspberry Pi 3 with two USB-Ethernet adapters as the OpenVSwitch, a Raspberry Pi 4 with an attached GPS/LoRa shield, and a Dragino LoRa gateway. In the raspberry Pis, I will install OpenVSwitch and I will configure it to listen to the Faucet controller[23]. In addition, I will deploy a backup controller, using the same configuration, in a remote location. Lastly, I will deploy three supportive virtual functions, a Dynamic Host Configuration Protocol(DHCP) server, a Network Address Translation(NAT), and BRO intrusion detection system(IDS), in the Raspberry Pi where the switch is installed. A graph of the topology is shown in figure 3. By using the controllers' features as well as TCPdump and Wireshark, I will gather the available information and evaluate if the SDN provides better visibility and control on the sensors. This experiment will give me an insight into the benefits or the drawbacks that an SDN-based solution has on a sensor network.

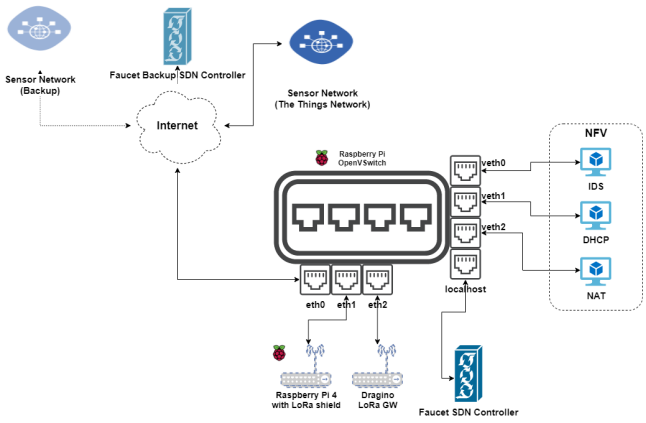


Figure 3: The topology of the experiment 1.

4.2 Switch failure experiment

For the second experiment I will create a redundant topology inside the network by using four virtual switches. One switch will provide a gateway to the Internet and the backbone of our infrastructure. On a different switch I will attach the LoRa gateway that listens to a LoRaWAN I/O controller that sends messages every 15 seconds. The two switches communicate using intermediate switches as can be seen in figure 4. After the communication is established, I will bring down on of the intermediate switches in order to replicate a sudden failure of equipment, and I will observe the behaviour of the system in such a situation.

This experiment will give us a better understanding on the response of the system in case of sudden failure, how it adapts, and how the centralized control of the SDN helps to mitigate this problem.

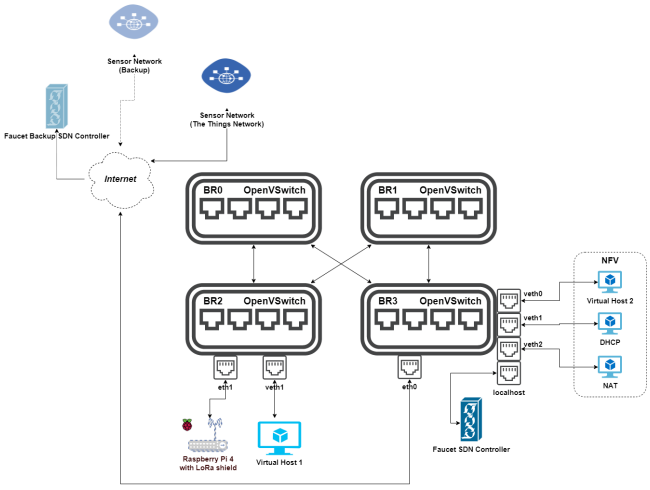


Figure 4: The redundant topology of the second experiment using four virtual switches.

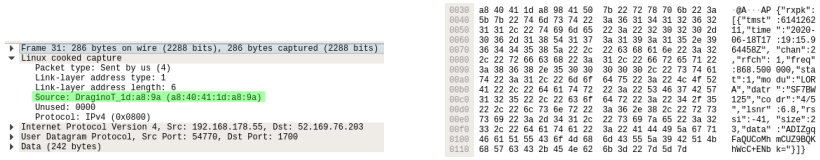
5 Results

Network control experiment

By centralizing the control of the network, changes over the topology were easy to apply and control by changing the configuration file(faucet.yaml) of the controller. The configuration was applied on the OVS immediately after I reloaded the controller, which in Raspberry Pi hardware took a few seconds.

Besides, I was able to block and redirect the traffic from the LoRa gateways by creating access lists (ACLs) and applied them to the switch. As the sensors are in range of both gateways, by blocking the flow from one gateway, I received and forwarded the sensor messages from the second gateway. This allowed me to deploy additional gateways in case of failure, and enable their traffic only if needed. Although in this case I only used one switch, it is trivial to apply the same ACL in all the switches of the network and replicate the same behavior. Moreover I can create ACLs to filter traffic that originates from our own sensor equipment and drop the rest. In a traditional network setup, this would be possible either by reconfiguring multiple switches or the LoRa gateways.

Additionally, LoRaWAN gateways operate entirely at the physical layer and, in essence, they are LoRa radio message forwarders. Consequently, I was not able to control the LoRa sensors individually on a network level. From the packet analysis, I could not distinguish individual nodes because both gateways overwrote the source address with the MAC address of the LoRa module, as shown in figure 5. Thus, I can distinguish from which gateway a LoRa packet originates, but I could not specify the actual sensor. A deeper analysis of the packet revealed that the data field of the packet contains enough information to uniquely identify a sensor. The actual data is encrypted, and the decryption



(a) MAC Address **(b)** Data Field

Figure 5: Left: Capture of a LoRa sensor packet using Wireshark. The destination IP is a The Things Network address. The source MAC address is the address of the Dragino gateway Right: The data field from the same packet.

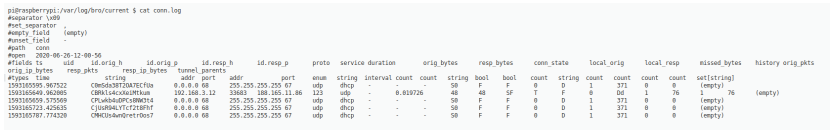
key is in the sensor network server. In a network with limited resources, such as a network that transmits over LTE channel[24], I can combine a virtual deep packet inspection firewall instance and filter the output traffic according to our needs. Although this solution requires hardware that is capable to inspect heavy loads of traffic, inside the infrastructure.

During the experiment, I disconnected the controllers in order to simulate the behavior of loss of connection with the controller. When I disconnected the first one, the OVS got its configuration file from the backup controller immediately. When the switch completely lost connection to the controllers, by default, it went to standalone mode which causes the datapath to act as an ordinary MAC-learning switch. The OVS daemon, named ovs-vswitchd, will continue to retry connecting to the controller in the background and, when the connection succeeds, it discontinues its standalone behavior. During this time, I was still able to receive LoRa messages in the network server. The log activity of the switch shows its attempts to reconnect to the controller, as shown in figure 6. An alternative to this behavior is the secure mode. In this case, the switch will not set up flows on its own when the controller connection fails.

Furthermore, it was possible to redirect the traffic from all the LoRa gateways to a secondary sensor network server in the backend of the infrastructure, by changing the configuration file of the controller. This allowed me to change or expand the setup or configure different sensor network servers for different gateways. It is possible to use the northbound API of the controller to automate this procedure [6][25] although due to time restrictions, I did not implement it. Furthermore, a virtualized load balancing function could redirect the network traffic to additional sensor servers when the load exceeds a specified level, or when the sensor network server is inaccessible for any reason.

In addition, I can forward the traffic from any port to an IDS system for inspection. I further investigated this case and I deployed one more virtual host, which runs a BRO instance, a passive, open-source network traffic analyzer. I changed the controller's configuration file to send the traffic to the IDS instance as well. A report from BRO with IP, TCP, UDP and ICMP connection details can be found on figure 6.

Lastly, OVS allows multiple ways for communication with the controller. By default the controller communicates with the switch over HTTP, which has no encryption. From my packet analysis I was able to recognize flows, routes, MAC addresses of source and destinations and more information that gives away information about the network topology, as shown in figure 7. Although this is not a secure configuration, OVS supports secure southbound communication



pi@raspberrypi:/var/log/bro/current $ cat conn.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2020-06-26-12-00-56
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto service duration orig_bytes resp_bytes conn_state local_orig local_resp missed_bytes history orig_pkts orig_ip_bytes resp_pkts resp_ip_bytes tunnel_parents
#types time string addr port addr port enum string interval count count string bool bool count string count count count count set[string]
1593165595.907522 CDmSdo3BTJ9A7ECfUa 0.0.0.0 68 255.255.255.255 67 udp dhcp - - - S0 F F 0 D 1 371 0 0 (empty)
1593165649.962085 CBRk14ccXoiMtkum 192.168.3.12 33603 188.165.11.86 123 udp - 0.019726 48 48 SF T F 0 Dd 1 76 1 76 (empty)
1593165659.575569 CPLwbXaUPCrbRNCt4 0.0.0.0 68 255.255.255.255 67 udp dhcp - - - S0 F F 0 D 1 371 0 0 (empty)
1593165723.425635 CjUsR94LYTcf2tBFhf 0.0.0.0 68 255.255.255.255 67 udp dhcp - - - S0 F F 0 D 1 371 0 0 (empty)
1593165787.774320 CMHCUs4wnQretrOos7 0.0.0.0 68 255.255.255.255 67 udp dhcp - - - S0 F F 0 D 1 371 0 0 (empty)

(a) conn.log

```
2020-06-26T12:12:27.540Z|00848|rconn|INFO|br0<->tcp:127.0.0.1:6653: connection closed by peer
2020-06-26T12:12:28.100Z|00849|rconn|INFO|br0<->tcp:127.0.0.1:6653: connecting...
2020-06-26T12:12:28.100Z|00850|rconn|WARN|br0<->tcp:127.0.0.1:6653: connection failed (Connection refused)
2020-06-26T12:12:28.100Z|00851|rconn|INFO|br0<->tcp:127.0.0.1:6653: waiting 2 seconds before reconnect
2020-06-26T12:12:30.100Z|00852|rconn|INFO|br0<->tcp:127.0.0.1:6653: connecting...
2020-06-26T12:12:30.100Z|00853|rconn|WARN|br0<->tcp:127.0.0.1:6653: connection failed (Connection refused)
2020-06-26T12:12:30.100Z|00854|rconn|INFO|br0<->tcp:127.0.0.1:6653: waiting 4 seconds before reconnect
2020-06-26T12:12:34.100Z|00855|rconn|INFO|br0<->tcp:127.0.0.1:6653: connecting...
2020-06-26T12:12:34.100Z|00856|rconn|WARN|br0<->tcp:127.0.0.1:6653: connection failed (Connection refused)
2020-06-26T12:12:34.100Z|00857|rconn|INFO|br0<->tcp:127.0.0.1:6653: continuing to retry connections in the background but suppressing further logging
2020-06-26T12:12:42.100Z|00858|rconn|WARN|br0<->tcp:127.0.0.1:6653: connection failed (Connection refused)
2020-06-26T12:12:50.100Z|00859|rconn|WARN|br0<->tcp:127.0.0.1:6653: connection failed (Connection refused)
```

(b) ovs-vswitchd.log

Figure 6: Up: Logs from the connection details captured from BRO on VLAN100. Down: When the OVS losses connection with the controller, it produces alerts.

with Transport Layer Security(TLS).

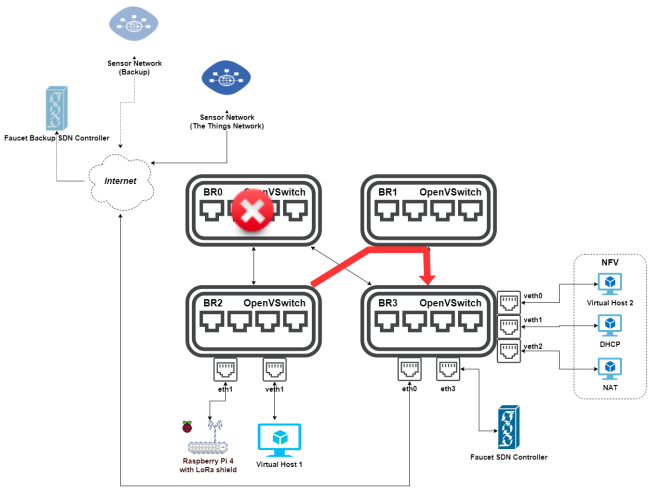
```
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="33:33:00:00:00:00/
ff:ff:00:00:00:00",inst_count="1",priority="8208",table_id="4",vlan="100",vlan_vid="4196"} 0.0
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="33:33:00:00:00:00/
ff:ff:00:00:00:00",inst_count="1",priority="8208",table_id="4",vlan="200",vlan_vid="4296"} 0.0
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="33:33:00:00:00:00/
ff:ff:00:00:00:00",inst_count="1",priority="8208",table_id="4",vlan="300",vlan_vid="4396"} 0.0
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="",inst_count="1",priority=
"8192",table_id="4",vlan="100",vlan_vid="4196"} 0.0
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="",inst_count="1",priority=
"8192",table_id="4",vlan="200",vlan_vid="4296"} 1.0
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="",inst_count="1",priority=
"8192",table_id="4",vlan="300",vlan_vid="4396"} 1.0
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="",inst_count="0",priority=
"0",table_id="4",vlan="",vlan_vid=""} 0.0
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="01:00:0c:cc:cc:cc",inst_co
unt="0",priority="8240",table_id="4",vlan="",vlan_vid=""} 0.0
flow_packet_count_flood{cookie="1524372928",dp_id="0x1",dp_name="sw1",eth_dst="01:00:0c:cc:cc:cd",inst_co
unt="0",priority="8240",table_id="4",vlan="",vlan_vid=""} 0.0
```

Figure 7: OpenFLow packet capture in plaintext.

Switch failure experiment

In this experiment, I took advantage of the controllers stacking mode. Stacking allows decisions such as switching and routing to be made by the main controller in the context of the whole network. First, I created four virtual switches, I assigned them ports and I connected them to the controller. After this point, the whole configuration of the network was done from the controller. This time, with multiple switches, the centralized control proved to be beneficial for troubleshooting the network.

A centralized log file with information about all the switches was created by the controller. When I introduced a sudden failure in the network, by bringing down BR0, I was able immediately recognize the source of failure on the network, by getting an alert, and the process of recovering from all the switches as you can see in figure 8. An example scenario, where this behaviour could be beneficial in a sensor network, is shown on figure 9, where I deployed a backup LoRa gateway and I disabled its flows to reduce traffic in the network. An application that uses the northbound API of the controller can receive the alert message from the controller when the primary gateway or switch fails, and take action by enabling the backup gateway flows. As mentioned in *Network control experiment*, due to time restrictions, I did not implement the controller handling application, but I tested this behaviour manually.



(a) BR0 is down

```
Jun 24 19:12:31 faucet.valve WARNING  DPID 1 (0x1) br0 datapath down
Jun 24 19:12:31 faucet.valve INFO     DPID 1 (0x1) br0 Port 1 (br0 stack link to br2) down
Jun 24 19:12:31 faucet.valve INFO     DPID 1 (0x1) br0 Port 2 (br0 stack link to br3) down
Jun 24 19:12:32 faucet.valve INFO     DPID 4 (0x4) br3 L2 learned on Port 8 e2:d0:5f:82:79:a9 (L2 type 0x0800, L2 dst fa:5c:33:de:6b:bd,
L3 src 192.168.2.18, L3 dst 192.168.2.2) Port 8 VLAN 200 (1 hosts total)
Jun 24 19:12:39 faucet.valve INFO     DPID 4 (0x4) br3 L2 learned on Port 7 de:57:a3:84:34:b4 (L2 type 0x86dd, L2 dst 33:33:00:00:00:02,
L3 src fe80::dc57:a3ff:fe84:34b4, L3 dst ff02::2) Port 7 VLAN 200 (2 hosts total)
Jun 24 19:12:39 faucet.valve INFO     DPID 4 (0x4) br3 L2 learned on Port 10 3a:92:da:38:50:88 (L2 type 0x86dd, L2 dst 33:33:00:00:00:02,
L3 src fe80::3892:daff:fe38:5088, L3 dst ff02::2) Port 10 VLAN 300 (4 hosts total)
Jun 24 19:12:40 faucet.valve INFO     DPID 1 (0x1) br0 LLDP for Port 1 inactive after 15s
Jun 24 19:12:40 faucet.valve INFO     DPID 1 (0x1) br0 LLDP for Port 2 inactive after 15s
Jun 24 19:12:40 faucet.valve ERROR    DPID 1 (0x1) br0 Stack Port 1 GONE, too many (3) packets lost, last received 15s ago
```

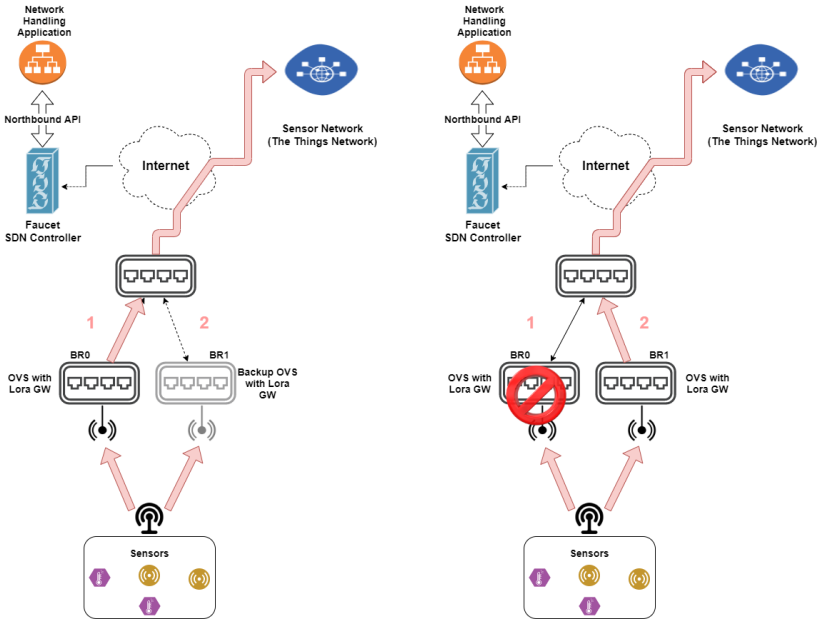
(b) Alerts on SDN controller

Figure 8: Up: The new flow, after I introduced a failure in BR0. Down: Logs from the SDN controller at the moment that i brought down the BR0 switch, which was the root switch in the configuration.

Lastly, the centralized control gives to the controller enough knowledge of the network to calculate a spanning tree for the network without the need for running a spanning tree protocol. After bringing down the switch BR0, from the topology shown in figure 4, the controller tries the same path for 15 seconds. When it fails to reconnect from the existing path, it calculates an alternative path which uses BR1. During this time contact with the sensors in the sensor network server was lost for 25 seconds. This means that we lost 1 or 2 messages per sensor, which transmit every 15 seconds. At the same time, I used iPerf3, a command-line tool used in diagnosing network speed issues, to exchange UDP traffic between virtual host1 and virtual host2. On average, packets were lost for 22 seconds, shown in figure 10.

6 Dicussion

In the experiments, I created a vendor-agnostic SDN environment and I tested different scenarios and functionalities on this network. These experiments showed that the reduced complexity from the centralized provisioning of the SDN can improve the control over the sensors. I was able to manage the traffic from individual flows, such as LoRa gateways, and set them different priorities and ACLs, and forward traffic from redundant gateways on demand. A use case would be a highly congested network where the traffic produced from the gateways where the critical sensors are connected, such as smoke or fire alarms, have higher



(a) Regular function **(b)** After switch failure

Figure 9: If a failure to a LoRa gateway, or switch occurs, the controller can enable an alternative flow from our backup gateways

priority over the rest of the network traffic. In addition, the knowledge of the complete topology from the controller allows for better mitigation in case of an unexpected hardware failure, as long as redundant communication to the controller is established. Even in a catastrophic scenario, we could still reprogram the network to forward the critical traffic from an out-of-band communication channel, until the damage is restored.
The centralized control of the network allowed me to forward or replicate the traffic to multiple services of the infrastructure. I could redirect the traffic to a different sensor network server, or load balance the traffic between multiple sensor network servers. The controller uses the northbound API to communicate with third party applications which can interfere with the network behaviour. Such functions could be load-balancers, firewalls, or sensor handling applications. In addition, NFV provides a flexible way to scale services according to our changing demands. Because the network functions are implemented in software, they can be easily moved to various locations in the network without having to install new equipment, which therefore can reduce the cost of the network and improve its reliability and security. I created an intrusion detection system in a virtualized host and I redirected the traffic to this host. At the same time, I examined the logs produced from different components of the network. Although this functionality is common in every network, the universal knowledge of the network makes it easier to monitor, and less prone to configuration mistakes. The benefits are more clear in bigger networks with more complex topology than those examined in this research. On the other hand, cases like the denial of service attack on OpenFlow[26], show that additional protocols on

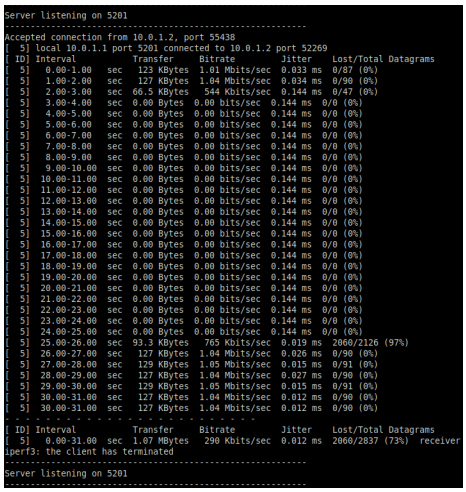


Figure 10: The results from iPerf3 test. After two seconds, I disconnected the switch BR0 and the messages got lost for 22 seconds before an alternative path is chosen.

the network stack, such as OpenFlow in our case, can increase the attack surface of the network. Additionally, compromise of the SDN controller, can lead to a complete compromise of the network, as it controls most of its components.

7 Conclusion

To substantiate my understanding, I created scenarios where unexpected failures happen in the networks, such as the loss of the network controller or a sudden hardware failure. Additionally I could redirect the traffic to different sensor network servers, in case of server failure. These scenarios helped me answer the following question. What benefits can an SDN-based architecture have as it concerns the redundancy of the system? The centralized control of the network, in combination with the fully digital nature of the system, makes handling redundancy and back up much easier. In addition, without the need for proprietary components, we can reduce the cost of the hardware and create more redundant and cost-effective solutions. In these scenarios, I used NFV to implement functionality such as the DHCP and NAT, that would traditionally be handled from the router. At the same time, I redirected the network traffic to alternative sensor networks servers, and load balanced the trafic between multiple instances. I, therefore, was able to answer the following question. What benefits can an SDN-based architecture have regarding the scalability of the system? SDNs combined with NFV allow to scale the infrastructure according to our needs. In addition, northbound APIs allow the automation of this procedure from application servers, and as a result, improve the scalability of the system. Lastly, I created ACLs that block traffic according to the needs of the network and forwarded the traffic to an IDS. These experiments helped me answer our third sub-question; What benefits can an SDN-based architecture have as it concerns the security of the system? Although the use of software-defined networks does not increase the security of the network per se, the universal knowledge of the network topology in combination with its centralized control, can improve

monitoring, and thus, improve our capabilities on detecting possible threats in the network and mitigate their effect. While this research provides deductive results, the experiments and background give me an insight into the effects of software-defined networks in critical infrastructure, which leads me to believe that they can benefit both the redundancy of the communication as well as its security.

8 Future Work

Software-defined networks are gaining popularity over the last years and their market share is constantly increasing. Future research should be aimed to the development of NFV that will manage sensor networks, giving emphasis in security and scalability of those networks.
Additionally, it would be beneficial to explore the benefits the P4 language, a language for programming the data plane of network devices, could have in a sensor networks, how it would increase the control of those networks, and finally what would the impact be on the security of the communication.

References

[1] L. Buttyan, D. Gessner, A. Hessler, and P. Langendoerfer, "Application of wireless sensor networks in critical infrastructure protection: challenges and design options," *Ieee Wireless Communications*, vol. 17, no. 5, pp. 44–49, 2010.

[2] S. B. Qaisar, S. Ali, and E. A. Felemban, "Wireless sensor networks in next generation communication infrastructure: Vision and challenges," pp. 790–803, 2014.

[3] A. Leon-Garcia, P. Ashwood-Smith, and Y. Ganjali, "Software Defined Networks," *Computer Networks*, vol. 92, pp. 209–210, 2015.

[4] S. Y. Zhu, S. Scott-Hayward, L. Jacquin, and R. Hill, Eds., *Guide to Security in SDN and NFV Challenges, Opportunities, and Applications* , 1st ed., ser. Computer Communications and Networks. Cham: Springer International Publishing, 2017.

[5] P. Goransson, *Software defined networks : a comprehensive approach* , C. Black, Ed. Waltham, [Massachusetts: Elsevier, 2014.

[6] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," 2014.

[7] K. P. E. Haleplidis, " Software-Defined Networking (SDN): Layers and Architecture Terminology," vol. 35, 2015.

[8] ETSI, "Network Functions Virtualisation Whitepaper #3," *Computer Networks*, vol. 20, 2014.

[9] J. Zheng, *Wireless sensor networks : a networking perspective* , A. Jamalipour and J. Zheng, Eds. Piscataway, New Jersey: IEEE, 2009.

[10] E. Buchmann, A. Kanzaki, and V. I. Zadorozhny, "International Workshop on Sensor Networks Technologies for Information Explosion Era (SeNTIE 2010): Preface," p. xxxvi, 2010.

[11] A. M. Ahmed, A. H. Abdalla, and I. El-Azhary, "Gateway placement approaches in Wireless Mesh Network: Study survey," pp. 545–547, 2013.

[12] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, "Wireless Mesh Software Defined Networks (wmSDN)," pp. 89–95, 2013.

[13] S. Salsano, G. Siracusano, A. Detti, C. Pisa, P. L. Ventre, and N. Blefari-Melazzi, "Controller selection in a Wireless Mesh SDN under network partitioning and merging scenarios," 2014.

[14] "Security of Software Defined Networks: A survey," *Computers and Security*, vol. 53, pp. 79–108, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016740481500070X

[15] M. Labraoui, M. Boc, and A. Fladenmuller, "Self-configuration mechanisms for SDN deployment in Wireless Mesh Networks," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2017, pp. 1–4.

[16] Z. Zhang, Z. Zhang, R. Wang, Z. Jia, H. Lei, and X. Cai, "ESD-WSN: An Efficient SDN-Based Wireless Sensor Network Architecture for IoT Applications," in *Algorithms and Architectures for Parallel Processing*, S. Ibrahim, K.-K. R. Choo, Z. Yan, and W. Pedrycz, Eds. Cham: Springer International Publishing, 2017, pp. 735–745.

[17] R. Kloti, V. Kotronis, and P. Smith, "OpenFlow: A security analysis," pp. 1–6, 2013.

[18] S. Seeber, G. D. Rodosek, G. Hurel, and R. Badonnel, "Analysis and Evaluation of OpenFlow Message Usage for Security Applications," pp. 84–97, 2016.

[19] E. S. Farrell, "Low-Power Wide Area Network (LPWAN) Overview," vol. 43, 2018.

[20] "Openwrt firmware," https://openwrt.org/, July 2020.

[21] L. Foundation, "Openvswitch," https://www.openvswitch.org/, June 2020.

[22] "Faucet controller," https://faucet.nz/, June 2020.

[23] F. foundation, *Faucet controller*, 2020 (accessed July 2, 2020). [Online]. Available: https://faucet.nz/

[24] J. Markkula and J. Haapola, "LTE and hybrid sensor-LTE network performances in smart grid demand response scenarios," pp. 187–192, 2013.

[25] C. Rocha Vasconcelos, R. C. M. Gomes, A. F. B. F. Costa, and D. Dias C. Da Silva, "Enabling high-level network programming: A northbound API for Software-Defined Networks," pp. 662–667, 2017.

[26] S. Hommes, R. State, and T. Engel, "Implications and detection of DoS attacks in OpenFlow-based networks," pp. 537–543, 2014.