SOFTWARE-DEFINED WIDE-AREA NETWORKS FOR DISTRIBUTED MICROGRID POWER SYSTEMS

BY

XUANYAO ZHANG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Advisor:

Research Assistant Professor Sibin Mohan

Abstract

Cyber-physical systems have increasingly taken advantage of packet-switching networks for control and data acquisition. A major example is the realization of the smart grid. On the networking side, software-defined networking (SDN) has been trending for the past decade. With the help of SDN, we are moving towards power grids that have both intelligence and security. In this thesis, we focus on providing a versatile SDN infrastructure for power-system applications in the environment of microgrids. We conduct simulations and collect statistics to demonstrate that the SDN approach facilitates communications and enhances security for certain microgrid applications.

Acknowledgments

I would like to first express my gratitude to Professor Sibin Mohan, for his guidance on this thesis and his advice on career development. I also want to thank Ashish Kashinath from our SyNeRCyS research group with whom I have had the pleasure to work and, whenever I was bored, to chat. I could always turn to him for help whenever I was in need.

I would also like to express my gratitude to researchers Alfonso Valdes and Richard Macwan from the Information Trust Institute, with whom I often worked until afterhours. Last but not least, I thank the rest of the team working on the project for Cybersecurity for Energy Delivery Systems (CEDS), including Dmitry Ishchenko from ABB and David Lawrence and Stuart Laval from Duke Energy. They expanded my knowledge beyond my discipline and provided me the opportunity to work on a project that ultimately resulted in this thesis.

Table of Contents

Chapter 1

Introduction

In the electric power industry, the trend is toward smartness for better control and smallness for better efficiency and resiliency. This trend has led to the concept of microgrid [1]. A microgrid is a localized group of electricity sources and loads within clearly defined electrical boundaries. A microgrid as an entity can be connected to a main grid and other microgrids in grid-connected mode, trading energy deficit or surplus and balancing the load on a larger scale. Or it can be isolated from others and operate entirely on its own energy resources, thus preventing the spread of grid failure due to natural disasters or human attacks [2]. The communication side largely follows the topology of a microgrid design, i.e., intra- and inter-microgrid communications.

A very simple microgrid may consist of a microgrid controller, a human-machine interface, a battery system and some protective relays. SCADA (supervisory control and data acquisition) protocols have evolved in the past four decades from serial protocols such as Modbus [3] to modern standards that use switched networks, including Ethernet and TCP or UDP over IP. Examples of these include IEC 61850 [4] and DNP3 [5]. SCADA operation consists of a master that polls devices for measurements, performs calculations and possibly issues commands to devices capable of undertaking control actions. The actions, for example, can be tripping a relay to isolate a fault, dispatching distributed energy resources (DER) to balance the load with the main grid and other microgrids or changing DER power input settings in response to conditions.

As the electric power sector increasingly adopts smart-grid technology, network-enabled devices (for measurements) and control have become widespread. Reliance on common lower-layer standards such as Ethernet and IP enables greater versatility in configuration, topology and coordination for devices. However, it also poses challenges in connectivity and security as newer communication protocols emerge (such as OpenFMB [6]) and the attack surface increases as evidenced in the recent cyberattack on Ukraine's power grid [7]. Such trends in power systems prompt this research to leverage a new networking paradigm, software-defined networking (SDN) [8], to better serve the infrastructural need of power system networks.

Conventional networks work in a distributed fashion where routers use well-defined protocols to exchange messages with peers to converge to a steady state so that packets are

correctly forwarded. SDN eliminates this process by having a central authority that controls routers directly. This paradigm allows the central entity to have a global view of the networks. Therefore, it can swiftly respond to any change of state and enforce new policies. We find this approach distinctly suitable for managing the networking infrastructure of power systems containing microgrids. With SDN, we can achieve faster routing, traffic prioritization and device or microgrid isolation. These qualities are well suited for microgrid designs because they are real-time systems where the speed of communication needs to match up to that of the change of electrical characteristics.

The typical networking setup of power systems retains conventional techniques, including the use of point-to-point VPNs to bridge field networks with central office, along with standard routing protocols. This setup only provides slow routing, sub-optimal routes and coarse-grained protection against compromised components. This research, however, aims to overcome the drawbacks of conventional networks by using SDN. In this thesis, we intend to demonstrate the possibility that the SDN approach can (a) facilitate device communication of OpenFMB-DDS applications across large geographical distances, (b) improve the performance in latency for DDS applications compared to some existing solution and (c) enhance security of the power system with fault isolation.

The rest of this thesis is organized as follows: In Chapter 2 , we introduce OpenFMB-DDS, a new framework we seek to support and tools we use to achieve such goal. In Chapter 3 , we review past research where SDN and power grids intersect and some existing solutions, including commercial ones. In Chapter 4 , we present the assumptions, the problem statement, the approaches and the results. In Chapter 5 , we present a physical setup to further prove the viability of our solution. In Chapter 6 , we discuss possible alternative approaches. Lastly, in Chapter 7 , we propose some possible future development, reflect on the entire process of our design and finally conclude this thesis.

Chapter 2

Background

In section 2.1, we explain the advantages of a microgrid design and a sample topology. In section 2.2, we discuss the need for an open standard for future devices and discuss how OpenFMB charts a course forward. In section 2.3, we provide a high-level view of the reliable middleware that bridges OpenFMB and the underlying network. The two sections provide us some background knowledge of smart-grid designs and the motivation for this research. Then in section 2.4, we introduce the concept of software-defined networking and discuss why it helps achieve the goal of the research. Lastly in section 2.5, we prepare readers with the knowledge of the tools and protocols in use.

2.1 Microgrid

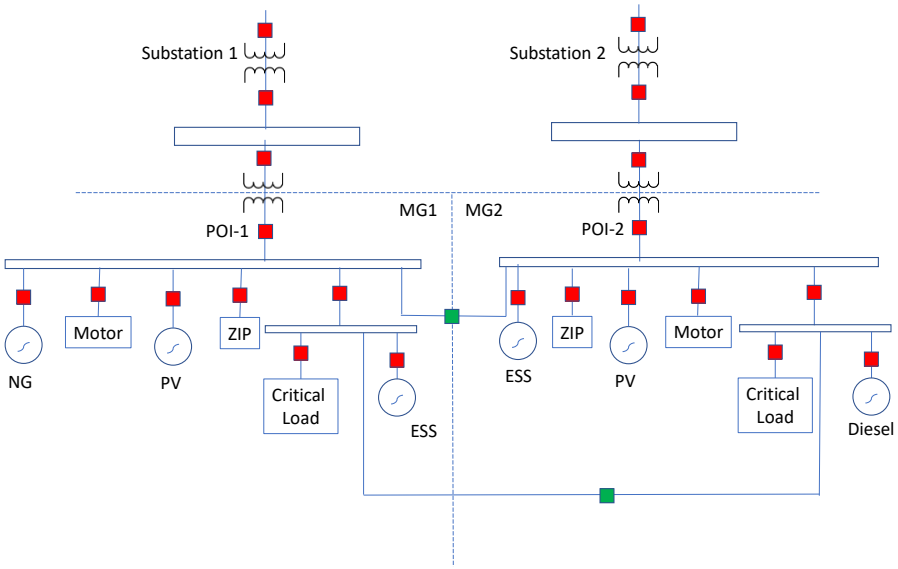

Figure 1 A use case with two microgrids. The red and blue cubes represent reclosers, with red ones being closed and green ones being open by default. Circular devices represent power sources and rectangular ones power sinks. Lines and bars represent electrical interconnections. Abbreviations are as follows: MG: Microgrid; POI: Point of Interconnection; NG: Natural Gas; PV: Photovoltaic Panel; ESS: Energy Storage System; ZIP: Constant Impedance (Z), Current (I) and Power (P).
Diagram courtesy of Alfonso Valdes.

A traditional power grid design has a strictly hierarchical topology featuring centralized power plants, long-distance transmission lines and substations owned by utility companies. Central plants can take advantage of economies of scale, but this advantage has been offset by distributed

energy generation with the advent of smaller power sources (microsources). A microgrid design takes advantage of the proximity to residential, commercial and industrial areas to achieve greater efficiency and flexibility. Such design, with advanced controls, could possibly reduce the loss in distant power transmission, reduce disturbances in terms of power quality, provide resiliency in the event of outage [9] and increase efficiency through the use of waste heat [1].

An example use case is demonstrated in **Figure 1**. It has two microgrids, MG1 and MG2, that are electrically connected to each other and, through substations, to the main grid. MG1 and MG2 maintain electrical connection to the main grid but electrical isolation between themselves by default. They can dynamically reconfigure electrical connections by using the reclosers in response to certain conditions.

2.2 The Open Field Message Bus

Unless one chooses to purchase power devices from a single vendor, there is a high likelihood that designers will encounter heterogeneous devices and different protocols. Traditional protocols are usually single-purpose that use a centralized control where there is clear dichotomy of masters and slaves. A conventional setup is to connect the field devices to the utility central office one by one (point to point). The centralized entity is responsible for collecting telemetric data from field devices for processing and issuing commands back to them. This, however, does not provide interoperability because field devices cannot directly interact among themselves.

The Open Field Message Bus (OpenFMB) [6] seeks to address these issues by putting forth an open-source common information model that every field device shall conform to, either as is for newer devices right off the shelf or through adapters for existing ones. This new model enables distributed intelligence and scalability for deployment. Additionally, due to its distributed nature, more robustness in connectivity and autonomous decision making can be achieved.

2.3 Reliable Middleware

OpenFMB's versatility comes not only from its data model (the upper layer) but also from the variety of middleware it can be adapted to, such as Message Queue Telemetry Transport (MQTT), Advanced Message Queue Protocol (AMQP) and Data Distributed Service (DDS) [10]. In this section, we introduce DDS and explain why we focus on this middleware over the others.

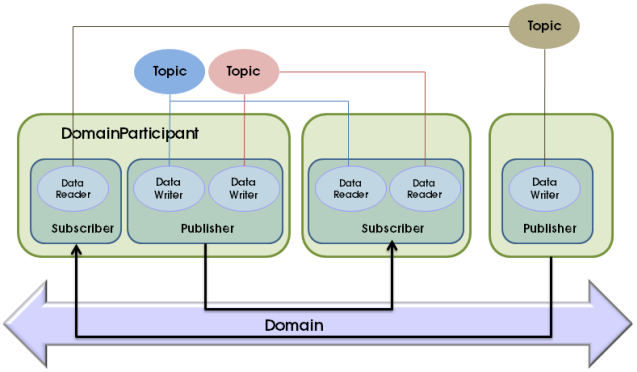

Figure 2 Logical view of DDS components [11]. The **domain** here facilitates logical isolation of different domain participants**. Domain participants** associate with each other via **topics**.

The aforementioned protocols conform to the publish-subscribe model. Unlike a TCP/IP connection where end hosts are identified by their respective IP addresses and port numbers, in these protocols, end hosts exchange messages identified by their topics via the middleware. For example, in a vehicular system consisting of wheel speed sensors (WSS), anti-lock braking system (ABS) and electronic stability control (ESC), both ABS and ESC will be interested in the speeds of the wheels, so both naturally subscribe to the speed data published by the WSS. In a publish-subscribe model as shown in **Figure 2**, data are not explicitly sent or retrieved from one endpoint to another but rather through topics that the endpoints share interest in. In this case, the topic is "Speed".

As the name suggests, DDS works in a distributed fashion. Unlike MQTT or AMQP, it does not rely on a broker server that can be a single point of failure. The DDS framework [12] is further divided into the data model and the wire protocol (Real-Time Publish-Subscribe protocol or RTPS). DDS allows users to define their own data objects and message filtering based on the attribute-value pairs in those objects. We avoid discussing the object model since this is largely handled by OpenFMB. Additionally, as this is a thesis on networking, our focus is limited to the wire protocol and how it interacts with our networking logic.

2.4 Software-Defined Networking

Software-defined networking (SDN) [8] has been a trending topic since its inception a decade ago. In this section we introduce what it is and how it facilitates communication and provides security.

Traditionally, core network switches use distributed protocols such as OSPF and BGP [13] to learn the global topology and peer relationships. They then calculate the optimal paths to destinations from the learned topology or relationships in order to determine which neighbors should be forwarded the packets, subject to business relationships and regulations. Due to the nature of these distributed algorithms, it is a slow process for the switches to converge to a consistent state where packets are correctly and efficiently forwarded. Furthermore, any change of state in the topology (such as a switch breaking down or a link becoming congested) will disrupt the equilibrium and cause the (slow) convergence process to start again.

SDN seeks to eliminate the need for a distributed process by introducing a centralized control plane as a replacement for the routing logic residing at individual switches. The controller gathers routing messages from border gateway routers, computes the optimal solutions and then formulates them as simple match-action rules to install onto the switches (forwarding plane). A full-scope implementation also has an application layer that interfaces with the controller to allow the user to change switching behaviors and monitor the state of the network. SDN speeds up routing, eases management and provides possibility for traffic engineering that cannot be achieved in a traditional network.

This thesis focuses on a specific application of SDN, viz., the software-defined wide-area network (SD-WAN) because our efforts aim primarily to provide connectivity for field devices within microgrids that constitute local-area networks (LAN) and across multiple microgrids via a wide-area network (WAN). We also provide support for isolating individual devices or entire microgrids to protect the power system from compromised nodes.

2.5 Tools and Protocols

Mininet [14] is an emulator that allows rapid prototyping of a large network. It leverages the Linux kernel to create virtual Ethernet interfaces (veth) in different network namespaces and Open vSwitch to create virtual switches that the interfaces can attach to. With veth comes the benefit of capturing and analyzing traffic with tools such as Tcpdump and Wireshark [15]. Mininet also provides shell interfaces (bash) for virtual hosts through process-based virtualization so that traffic can be generated and received in the simulation, thus providing an end-to-end testbed for SDN implementations. Most of the testing in this thesis was conducted in the Mininet framework.

Open vSwitch [16] (OVS) provides the functionality of the SDN forwarding plane (switches and links). It consists of several user-space utilities to configure topology, provide OpenFlow interface for the SDN controller and interact with its kernel Datapath module that provides low-latency, high-throughput packet forwarding. As we move towards deployment, we use OVS commands to manipulate the OVS database directly to fine-tune and retain the setup in a persistent manner.

Ryu [17] is an open-source Python framework for developing SDN controllers. It supports OpenFlow protocol versions 1.0 through 1.5, the latest release. It also incorporates a Web Server Gateway Interface (WSGI) component for developers to implement the application layer of the SDN, enabling on-the-fly reconfiguration of the control plane.

OpenFlow [18] has emerged as the de-facto standard protocol for SDN development. It allows the controller to direct actions for packets based on their headers and provides a few methods to query for the switches' statistics, such as the number of bytes or packets that have been transmitted or received for a given interface and the state of interfaces.

2.5.1 IP Multicast

In the world of power systems [19], multicast plays a critical role for many publish-subscribe protocols. Many field devices do not have a great amount of computing power. The industrial Ethernet switches do not have large throughput either unlike commercial switches at data centers. Multicast can reduce unnecessary traffic by pushing traffic duplication toward the last links as far as possible, which, in turn, reduces the load on the senders and backbone switches.

IP Multicast is divided into two parts, the WAN segment and the LAN segment. The WAN side consists of routers that usually run Protocol Independent Multicast (PIM) protocols for forwarding traffic. PIM has two modes: the dense mode (DM) [20] and sparse mode (SM) [21]. The DM postulates a scenario where the majority of end hosts in the network are interested in certain particular multicast traffic, so the idea is to (a) flood the entire autonomous system with such traffic and (b) let downstream routers notify upstream routers that they do not want such traffic if there are no interested listeners. The SM does the opposite. It works by designating a rendezvous point (RP), where the senders and receivers "meet" to complete forwarding paths and multicast traffic is duplicated. Once the edge routers for listeners learn the sender's address, the routers can trace back and pull multicast traffic along the shortest path while pruning the original

path. Both methods are slow and inefficient and no ISPs, to our knowledge, support multicast without a special agreement. To further complicate matters, our microgrids might belong to different internet providers. The LAN side only involves the edge routers and end hosts to which they are connected. Internet Group Management Protocol (IGMP) or Multicast Listener Discovery (MLD) [22], [23] messages are exchanged for membership management in LAN. Essentially, whenever a host wants to join or leave a multicast group, it sends out a multicast message destined for that specific group to report "joining" or "leaving", and then the edge router is responsible for picking up such reports to keep track of which end host is in which multicast group. In addition to self-reporting by end hosts, the edge routers are also responsible for periodically sending out IGMP queries for membership reports, lest any leave messages be lost, or hosts crash before sending such message.

Chapter 3

Related Work

Most of the SDN research in the area of power systems seems to have focused on single-grid use cases. Pfeiffenberger et al. [24] presented a way for efficient data delivery of multicast traffic with fault tolerance and with a focus on IEC 61850 traffic. Cahn et al. [25] tried to adapt SDN for the power grid as it transitions into smart grid, in order to provide better management, auto-configuration and security.

Other research [26], [27] primarily deals with several different field devices within the same power grid and seeks to identify and optimize real-time flows in such a system or to improve security through filtering. I would argue such efforts still fall within the scope of the generic application of SDN. To the best of our knowledge, ours seems to be the first to use SDN for traffic, both within and across microgrids.

There is a component to RTI's DDS framework called Routing Service [28]. It is an out-of-the-box solution to bridge DDS applications across different publish-subscribe domains and LANs. The idea is to have a service application running in every LAN that subscribes to all DDS messages. The services are preconfigured with their peers' IP addresses and listening ports, so that messages can traverse from one LAN to another through the services that act like proxy servers. This solves the connectivity issue but is very inefficient as messages do not go from source to destination directly. Furthermore, this solution is limited to DDS or perhaps only RTI's implementation of it, thus lacking interoperability.

For the commercial use, SD-WAN [29] is gradually replacing traditional services such as T-carrier and MPLS [30]. It is mostly used to optimize traffic for multi-homed offices and provide dedicated and secured connectivity between branch offices. Existing solutions include Cisco's iWAN [31] and Riverbed's SteelConnect [32]. While they likely already provide multicast support across branch offices, they might not allow customizability for power-system protocols on a per-flow basis.

Chapter 4

Design and Implementation

In this chapter, we begin by presenting a power-grid use case that reflects a real-world scenario. Then we delve into the technical details of IP Multicast and we explain the core design of the SDN controller that is the control logic and the forwarding rules derived from it. Finally, we present an interface for external inputs to change the controller's behaviors and retrieve the state of the network, thus demonstrating a full-stack solution in SDN.

4.1 System Model

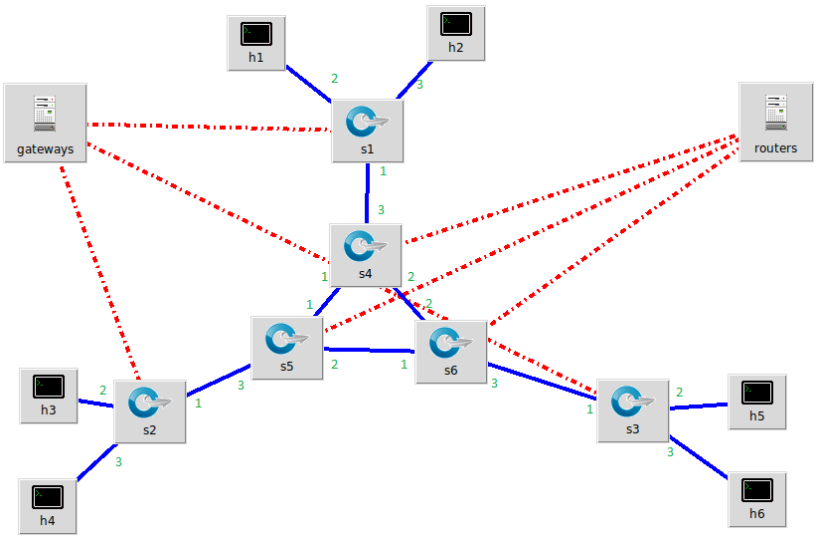

Figure 3 Topology for our experiments. "Gateways" and "routers" signify SDN controllers. Blue loops with arrows through them signify SDN switches. Command-line interfaces in black backgrounds signify end hosts. Links are in blue or red, respectively, representing the transport network and management network.

We simulate a physical setup that includes three microgrids inter-connected through a WAN and two field devices residing at each microgrid. We "own" every microgrid, e.g., the networking infrastructure and the field devices, but we do not have any control over the WAN that is operated by third parties such as AT&T and Verizon. **Figure 3** shows our simulated network in Mininet. Switches s1, s2 and s3 act as gateways, each residing at a microgrid, numbered from 1 to 3. We assume gateways have as many ports as we want so that all end hosts (field devices) can be directly connected to the gateways. This improves control as we can police the traffic of any given host with the controller for the gateways. Switches s4, s5 and s6 simulate the WAN. They have a separate controller because an energy company does not usually own the networking infrastructure

beyond their properties. Consequently, we do not assume any control over this controller. Following the conventional architecture of SDN, a separate management network is provided between the controllers and switches, connected through red links.

Each microgrid, also a subnet, is assigned a private IP range. A host thus can be assigned the address *172.16.<Grid ID>.<Host ID +1>*. The gateways themselves hold the addresses *172.16.<Grid ID>.1* and each of them also has a public IP or MPLS label.

4.2 Interacting with the End Hosts

A few commonplace mechanisms are implemented to handle protocols such as Address Resolution Protocol (ARP) [33] in order to provide standard IP and MAC unicast. We will not expand on these components in order to focus on the key issues.

4.3 The Control Plane

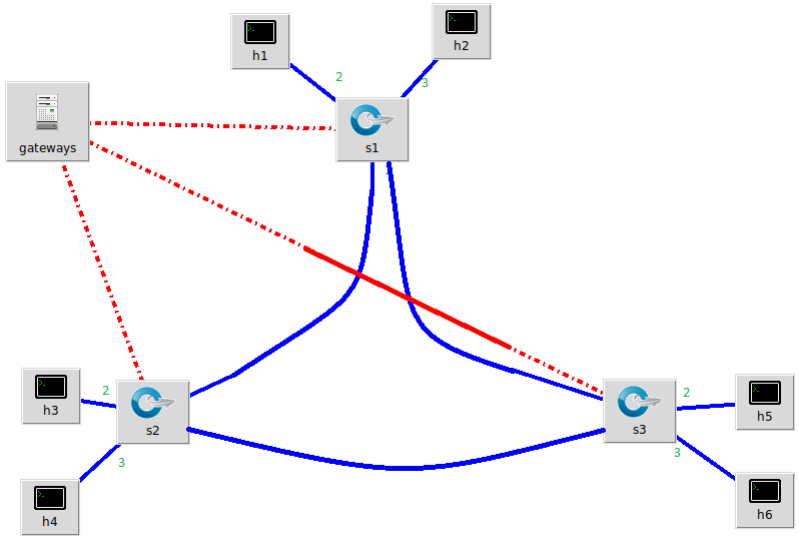

Figure 4 This is the topology of the overlay network formed using VXLAN. In this topology, there are three microgirds and correspondingly three gateways, each with two southbound ports to the end hosts and two northbound ports to other microgrids.

This design, as shown in **Figure 4**, seeks to flatten the design with the help of Virtual Extensible LAN (VXLAN), a layer-2 tunneling protocol [34]. A VXLAN end point takes the entire packet, encapsulates it in UDP/IP and sends it over to another end point. In this design, we set up as many northbound interfaces as there are peers, which means there will be $(n-1) * {}^{n}/_{2}$ point-to-point tunneling links, forming a full mesh. The first problem we encounter is loops. The traditional solution is the spanning tree algorithm that prunes additional links and makes them

inactive backup lest active links fail. This works fine in a small scale, physical LAN environment. In our case, however, each northbound link is potentially across a distant geography, with limited bandwidth and high latency. In order to preserve the optimal routes that are the direct links from one to another, we divide interfaces into two categories, the northbound ones that are VXLAN and the southbound ones that are end hosts (field devices). For broadcast traffic received from the southbound interfaces, a switch floods it to all other interfaces. For broadcast traffic received from the northbound interface, it only floods it to southbound interfaces, knowing they must have already been flooded to other switches by the first hop gateway.

4.3.1 Event Loops

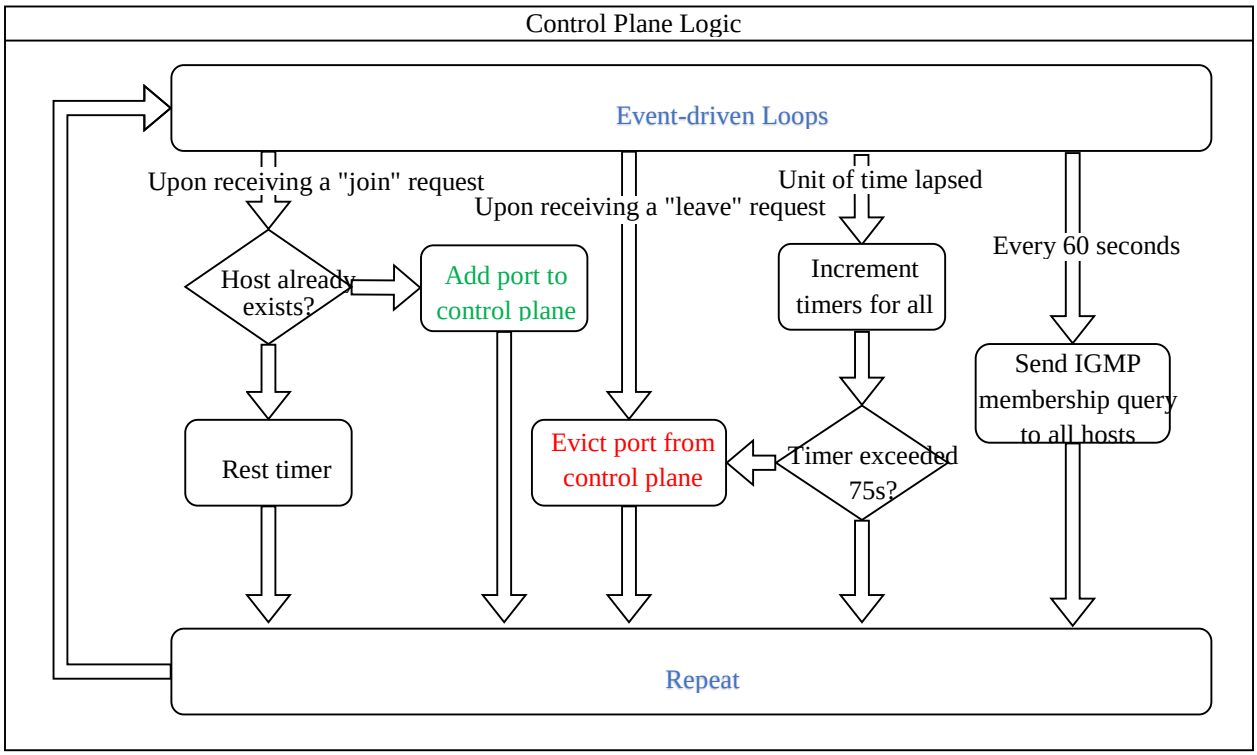

Figure 5 The control plane logic consists of four event loops. The rightmost loop is the query thread. The second from the right is the timer thread. The two loops on the left handle IGMP join and leave messages.

The control plane logic has multiple threads to deal with different aspects, as summarized in **Figure 5**. The query thread, every 60 seconds, sends out membership requests to southbound hosts, triggering passive membership reports within the response interval (10 seconds). The timer thread, every unit of time, increments the time counter for each listening port. If the time counter reaches 75 seconds (the amount of time should be greater than the query interval plus the response interval), it times out and is evicted from the control plane. A more frequent query results in a

higher-resolution view of the state of the network but increases the burden on the management network and the SDN controller. We choose an interval of 60 seconds which is about half of the suggested value of 125 according to the RFC [22].

The rest of **Figure 5** demonstrates the control plane logic for IGMP messages running for each gateway. Whenever an end host reports joining or leaving a multicast group, the messages are sent to the control plane logic at every gateway. If not already existent, the handling routine sets up a "listeners_port" dictionary for each multicast group and adds an entry with the key being the port number from which the report comes in and the value being an "IgmpListeners" class, as shown in **Figure 6**. Then, the address of the listener will be added to the set "listeners_addresses". Whenever a "join" message arrives from any end host, the "time_counter" is reset to zero for the listening port ("IgmpListeners") it is attached to. Whenever a "leave" message arrives, it removes such end host from the "listeners_addresses". There are two cases where an "IgmpListeners" class should terminate and evict itself, either when the timer reaches 75 or when all of its end hosts have reported leaving the group.

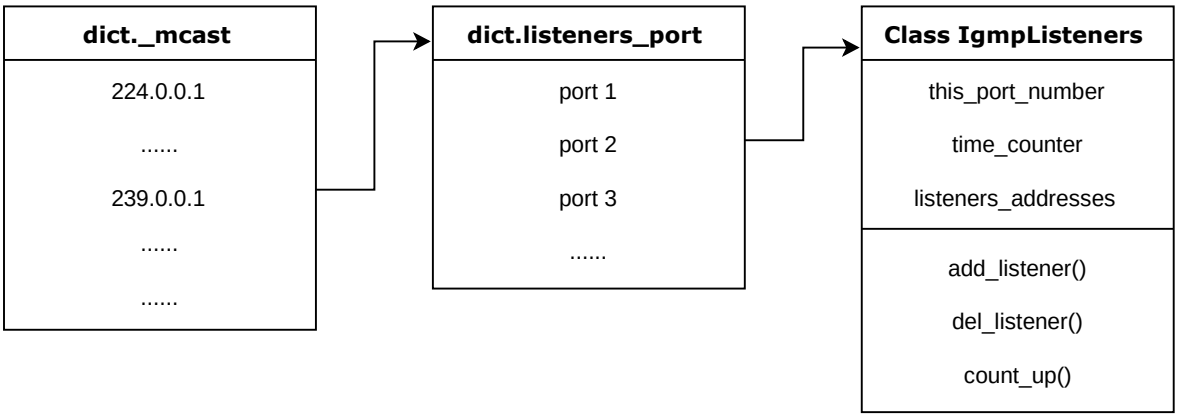

Figure 6 Data structure for bookkeeping. The "_mcast" structure is a dictionary that has multicast addresses as the keys and pointers as the values, pointing to other dictionaries that are "listeners_port". Each "listeners_port" dictionary has port numbers as keys and their values point towards "IgmpListeners" dictionaries.

4.4 Forwarding Plane

The data structure in **Figure 7** reflects what should be installed on the gateway switches. Each existing entry in the "_mcast" dictionary corresponds to two action groups that are collections of actions. They together have as many output ports as there are entries in the "listeners_port" dictionary where those ports are split into two groups based on locality, one for southbound interfaces and the other for northbound ones. The check for tunneling ID determines if a multicast

packet arrives from a remote microgrid (gateway), in which case the packet can only be duplicated for local ports (southbound) to prevent broadcast storm. Otherwise, a packet can be duplicated for any port as long as there is at least one listener attached, if such packet originates locally.

When an "IgmpListeners" class is evicted, it modifies the action group by removing the output port. If both action groups contain zero ports, meaning there are zero listeners anywhere, the flow rule is evicted from the switch.

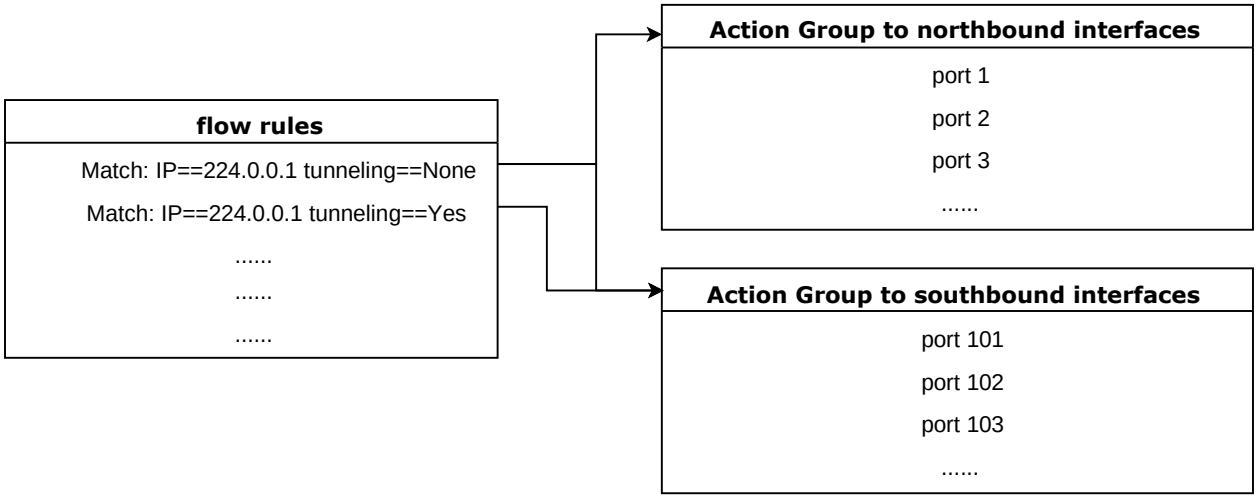

Figure 7 Forwarding plane structure showing a flow table for multicast packets and associated action groups. The flow rules table maintains two entries for each multicast group, one for packets from northbound ports and the other for from southbound ports. The entries in the action groups are ports the listeners are attached to. Ports in action groups should be separated based on northbound or soutbhound

Recounting the entries in the forwarding plane, for each IPv4 multicast group, there are two flow rules, two action groups and as many ports as needed in the two groups. In the OpenFlow 1.4 standard, there are 2^{32} action groups. In IPv4, there are 2^{28} multicast groups so we will not run out of action groups. But in IPv6, the number of multicast addresses far exceeds 2^{32}.

4.4.1 Simplify the Forwarding Plane Logic with MPLS

We soon realized some of the shortcomings from flattening the network into a giant LAN. The first problem comes from the fact that the number of tunneling links grow in the order of $O(n^2)$. This joint project is still in its early phase with the goal of deploying five microgrids. As the number of grids grows, however, this becomes increasingly unwieldy. The second issue comes from it being a LAN. VXLAN is a layer-2 tunneling protocol designed for building overlay networks for VMs in a data center. Messages like ARP are flooded across the entire overlay network, which puts too much burden on the network, especially the WAN part. The third problem

comes from dynamic configurability. OpenFlow is inherently incapable of processing any tunneling protocol headers, so we rely upon OVS to provide the tunneling end points. This requires reconfiguration for both the SDN controller and the gateways every time we want to add a new grid into the system, which is in the order of O(n).

With such consideration, we decided to switch to a design with the help of Multi-protocol Label Switching (MPLS). In this design, we restrict the LAN switching to within the microgrids, only allowing layer-3 traffic to go northbound. We incorporated proper ARP handling into the SDN controller to reduce the amount of traffic flooding. In addition, because MPLS is native to OpenFlow since version 1.1, we can reduce the northbound ports to only one per microgrid and migrate the dynamic reconfiguration of power grid topology to the SDN controller, such as adding a new microgrid to the existing setup. Now, the number of northbound links is reduced to the order of $O(n)$ and the number of gateways requiring reconfiguration to zero.

Flow rules
Match: IP==224.0.0.1 Actions={(Forward to port 1,2,3), (Push MPLS),(Forward to port 101)}
.....
.....
.....
.....

Figure 8 Forwarding plane structure showing the flow table for multicast packets in MPLS design, which only requires one entry per multicast group

Incoming traffic via the northbound interface is assumed to be always MPLS tagged, so it is first stripped of its MPLS header and then checked to see if it is a multicast packet before being sent to the multicast flow table. Another advantage in this design is that you do not need to differentiate the ingress port of a packet because the same action can be applied to all without causing broadcast storm, due to the fact that OpenFlow will ignore output actions to the ingress interface, unless explicitly specified. For the same reason, we no longer need to split every flow rule into two to separately handle traffic originating from the local grid versus traffic from remote grids, and we completely get rid of the use of action groups, resulting in similar flow rules laid out in **Figure 8**. This design greatly simplifies the forwarding plane and we are down to only one entry per multicast group to deal with on the forwarding plane.

4.5 The Application Layer

So far, we have dealt only with the forwarding plane and control plane. Now we need to introduce some human control on the fly. This is the application layer of the SDN hierarchy. We incorporate some Representational State Transfer (REST) interfaces into our controller to achieve some security measures as shown in **Table 1**.

Table 1 A List of APIs for the Application Layer

Name	*HTTP Request Path*	*Method(s)*	*Body of Response(s)*
	Description		
list_table	/gateways/table/{dpid}	GET	"ip to mac" "mac_to_port"
	Queries the controller for end hosts' information at a specific datapath. The response "ip_to_mac" shows the MAC addresses of end hosts with specific IPs. "mac_to_port" shows which southbound ports the end hosts with specific MAC addresses are attached to.		
isolate	/gateways/isolate/{dpid}/{port_no}	POST	"isolated_dpids_ports"
	Shuts down a specific port of a specific Datapath (gateway). If the "port_no" is 0, it shuts down every single port of the Datapath and cuts off any intra- and inter-grid communication. If the "port_no" is the northbound port, it shuts down the inter-grid communication but devices within the microgrid can still communicate with each other properly. Both actions will eliminate other microgrids' awareness of multicast state of the isolated microgrid, which, in turn, eliminates unnecessary northbound traffic that will eventually be rejected. If the "port_no" is one of the southbound ports, only that specific device is isolated from the rest of the system. The response "isolated_dpids_ports" keeps track of the isolated ports of the gateways.		
deisolate	/gateways/deisolate/{dpid}/{port_no}	POST	"isolated_dpids_ports"
	Undoes "isolate". The response is the same as that of the "isolate" API.		
allocate	/gateways/allocate/{dpid}	POST	None
	Allocates resources for a new microgrid that joins the system. The body of the POST method should contain the MPLS label assigned to this microgrid, the port number of the northbound interface, the IP and MAC of the default gateway and the subnet mask. If the inputs are correct and the resources to allocate pose no conflicts with existing setup, "200 OK" is returned. Otherwise it returns "400 Bad Request" or "409 Conflict".		

To use the APIs, one will send appropriate HTTP requests to the address *Domain Name or IP/HTTP Request Path*, with contents in braces replaced by actual identifiers. The responses are in JSON formats, returning a list of states in the controller. In the case of incorrect requests, a "404 Not Found" response is returned, unless specified otherwise.

Chapter 5

Deployment and Results

We implemented the setup described in Chapter 4 in hardware to demonstrate feasibility in a realistic scenario. However, we were met with multiple constraints. First, we do not have an MPLS infrastructure. Secondly, we could be limited to only one public interface per microgrid, forcing us to move forward with in-band SDN control, which means we will use the same network for transport and management. With such considerations, we try to emulate an MPLS infrastructure with the help of tunneling -- resulting in a mixed usage of VXLAN and MPLS.

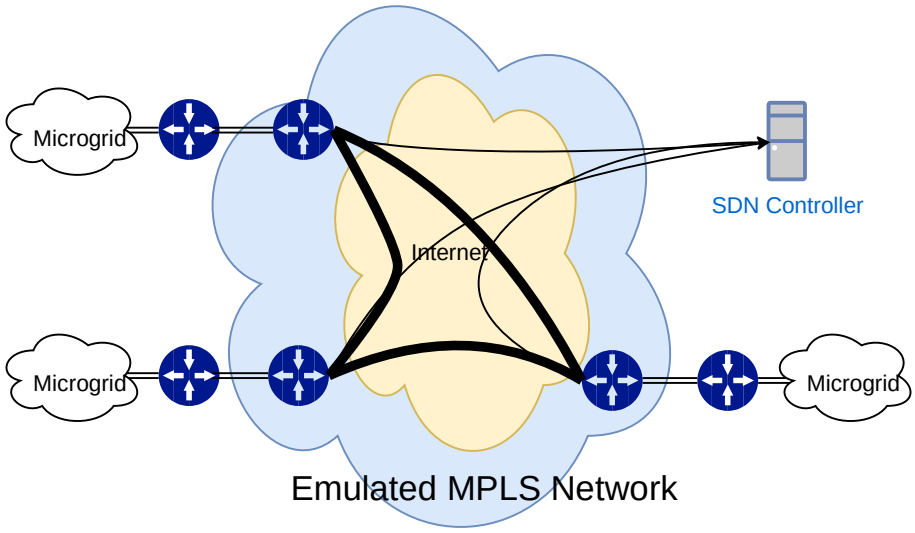

Figure 9 The current topology for deployment. The virtual routers on the blue cloud translate MPLS labels to VXLAN encapsulation. The routers outside the cloud are MPLS routers.

Figure 9 shows the high-level table-top setup of our experiment. The infrastructure is set up with three PC towers, one laptop and a simple layer-2 switch. Each PC tower has five Ethernet ports, with one connected to the layer-2 switch. The laptop is also connected to the switch and runs the Ryu SDN controller. Each of the three towers runs a Linux 4.15 with OVS 2.5 installed. It runs two instances, one being the MPLS SDN instance as described in section 4.4.1 and the other for converting MPLS-tagged packets to VXLAN encapsulation. The number of tunneling pairs still remains in the order of $O(n^2)$, but we have decoupled it from the control plane.

Most of the debugging was done using RTI's performance test (a command-line application to measure the latency and throughput in different configurable scenarios that use DDS middleware to send messages [35]). Therefore, any application built on top of DDS should work as is. But to move our design closer to reality, we also used the OpenFMB-DDS demo [36] to demonstrate the SD-WAN infrastructure. We have a total of four Raspberry Pis and five OpenFMB applications. We attach the Pis to two of the PC Towers, two for each. The applications running on the Pis are a human-machine interface (HMI), a battery simulator, a recloser simulator, a solar panel simulator and a load simulator. The latter four are assigned to the Pis, one for each, with the HMI assigned to the first Pi. The HMI, as shown in **Figure 10**, is a web UI that allows the user to monitor the state of the microgrid. If we isolate any port or grid, the associated end hosts will disappear from the HMI, indicating the communication channels to them have been cut off.

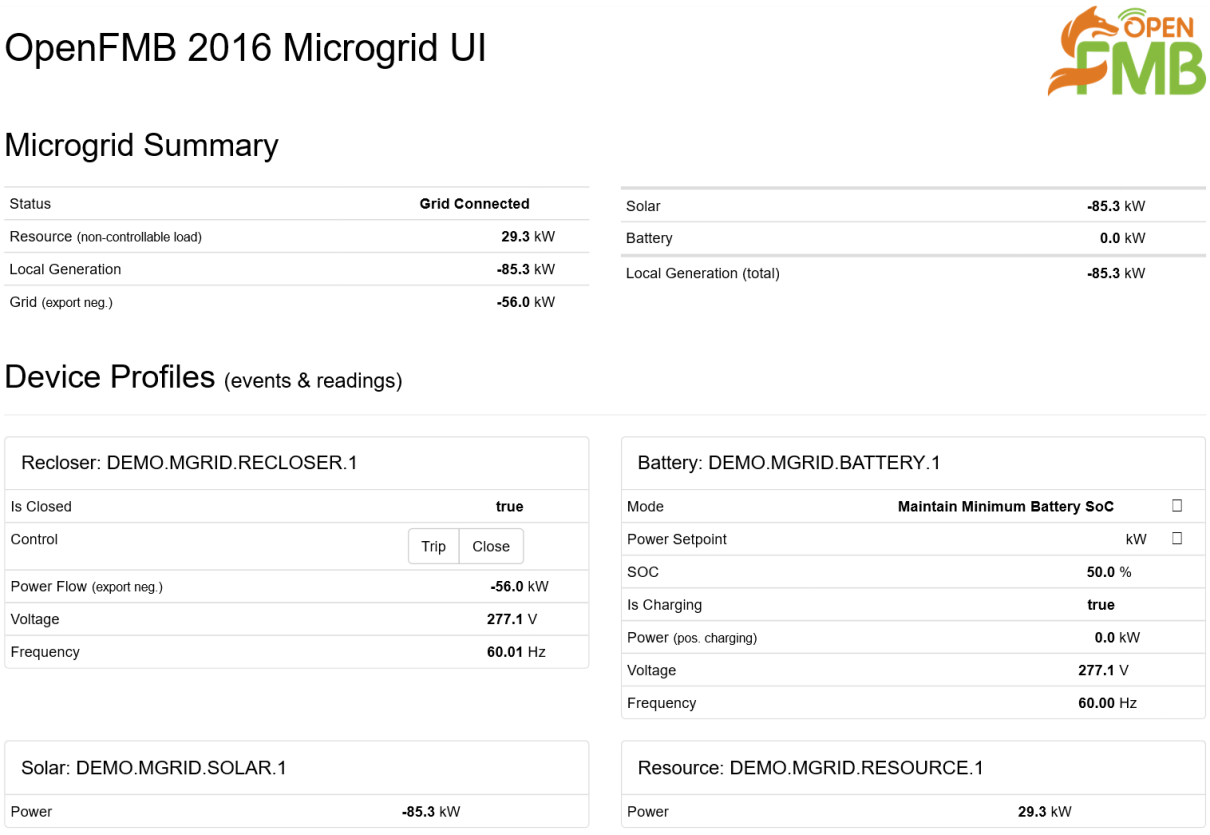

Figure 10 The HMI monitoring four field device simulators. This is a snapshot of the web UI. The Device Profiles section displays, clockwise from top left, a recloser, a battery, an electrical load, and a solar panel.

5.1 Results

Aside from the connectivity and security that have been tested, we can also compare our solution to existing ones. Centralized decision-making is faster than running distributed algorithms. This holds much more importance for multicast traffic due to the constant change of membership. Our design should be logically faster than any conventional routing protocols but comparing with the conventional designs would not be easy, because we neither know the exact WAN topology where we will deploy this solution nor have the commercial switches to construct a conventional network for such a topology. However, we can still compare it with an existing solution that seeks to avoid the problem of multicast across WAN, i.e., the RTI's Routing Service (RS). **Table 2** shows the results collected from running the latency test of RTI's Performance Test with the default QoS (Quality of Service) profile over a span of 600 seconds. The statistics were collected from two setups, one in an environment simulated with Mininet as described in section 4.1 and the other using the table-top setup in **Figure 9**. The test consists of (a) a publisher in one LAN that publishes data to test bandwidth and periodically inserts requests for pings in the packets and (b) a subscriber in another LAN that acknowledges the data and responds to the publisher for the ping requests. The test also includes an RS instance in the publisher's LAN for both setups but only the simulation in Mininet has another RS instance in the subscriber's LAN due to resource constraint. The results show that using RS incurs substantial latency compared to our solution (native connection). There are multiple reasons, but the first and foremost one is that the RS inserts one or more hops between the endpoints. The RS also uses its own QoS profile in data transport including messages parsing, filtering and reassembly. This additional packet processing contributes to the end-to-end latency. It should also be pointed out that the RS enforces the use of TCP/IP for reliable transport and NAT traversal across the WAN and the use of a reliable protocol might have negative impact on real-time applications.

Table 2 Latency Test in Different Environments

Latency (microsecond)		Avg	Std	Min	Max
Mininet	Native	85	27.1	42	8109
	RS (pub side only)	218	76.7	132	9906
	RS (pub and sub sides)	371	123.7	216	12491
Table-top	Native	888	169.6	482	9672
	RS (pub side only)	1431	255.0	1015	12513

Chapter 6

Other Design Considerations

6.1 Hub-and-Spoke Model with VXLAN

Constrained by the possibility that we will not have an MPLS infrastructure, we try to address the scalability issue in point-to-point tunneling. A solution could be to have a hub node residing independently of all microgrids and have all microgrids connect to it through tunneling as shown in **Figure 11**. This way, we keep the number of tunnels in the order of $O(n)$. However, aside from being similar to the conventional setup where the tunneling hub resembles the control room (central office), this approach has other obvious downsides. First, all inter-grid traffic will cross the hub thus severely burdening the hub and relevant links. Secondly, every inter-grid packet will take a detour through the hub resulting in causing unnecessarily latency, which is very undesirable for a real-time system. For these reasons, we chose not to proceed with the hub-and-spoke model.

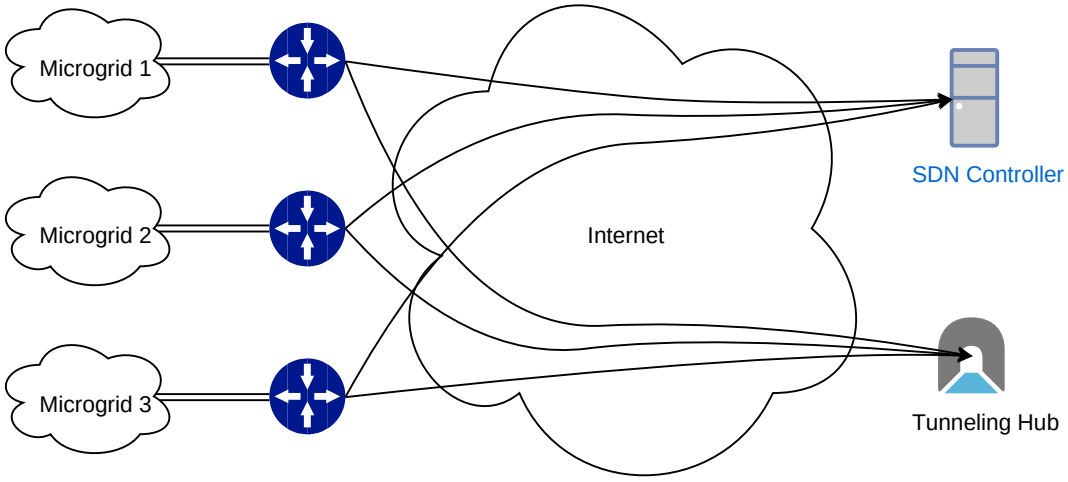

Figure 11 A possible hub-and-spoke model where each microgrid only maintains a single tunnel to the hub.

6.2 Building our Own SDN Switches

Having 5 ports per microgrid is enough for the time being, but as we progress further, we will attach more devices. The use case in **Figure 1** requires eight southbound ports per microgrid that can be satisfied with a regular PC tower with two 4-port network interface cards (NIC). However, inserting ever more NICs is not a scalable solution.

Inspired by the solution of separating the tunneling switch from the SDN switch, we are working on expanding the number of physical interfaces with the help of Virtual LAN (VLAN). We can create as many virtual interfaces as we need for a single port on the PC and assign each a unique VLAN ID. Secondly, we assign these virtual interfaces to the SDN switch where they appear to be physical interfaces from the SDN controller's perspective. Next, we designate the same number of untagged VLAN ports on the switch and a tagged trunk port and then bridge the untagged ports with the trunk port. Lastly, we connect the PC's physical port to the trunk port of the switch. Hence, the virtual interfaces of the SDN switch will mirror those untagged ports on the physical switch. This greatly expands the number of ports we need with the penalty of forcing every packet through the PC and congesting the trunk link.

So far, we have tested this approach for intra-microgrid (LAN) communication and collected some statistics with two Raspberry Pis (version 3 model B+). The networking infrastructure is built with an HP 2920-24G switch and a PC tower with a multiport NIC (OS: Linux 4.15 CPU: Intel Core i5-2500 @ 3.30Ghz; NIC: Intel 82576). **Table 3** shows the results from different types of connection between the two Pis by pinging from one Pi to another using default parameters over a span of 600 seconds. The results show OVS with VLAN expansion incurs about 0.1-0.3 milliseconds of latency compared to simpler configurations. Put in perspective, such latency penalty can be considered tolerable when compared to the end-to-end timing requirement of 4 milliseconds for a layer-2 substation protocol IEC 61850 GOOSE [37].

Table 3 Latency Statistics for Different Configurations

Latency (milisecond)	Min	Avg	Max	Std
Via only an Ethernet cable (direct link)	0.388	0.498	0.587	0.045
Via the switch (packets processed by hardware)	0.326	0.433	0.560	0.052
Via the PC (using two ports, packets processed by OVS)	0.399	0.612	0.841	0.054
Via the switch and the PC (using one port with VLAN, packets processed by OVS)	0.491	0.745	0.887	0.075

Chapter 7

Conclusions

7.1 Future Work

We have increased our attack surface by using a full-fledged operating system and in-band control for the gateways. Some mitigations include securing OpenFlow with TLS, encrypting tunneling traffic with IPSec [38] and hardening Linux with SELinux [39] and firewalling

There is some correlation between DDS' domain and topic and IP's multicast group. We may be able to achieve finer-grained control of topics by isolating a host from certain multicast groups such that a compromised or malicious host cannot subscribe to certain topics. We can also integrate an intrusion detection system to our system, such as Zeek (formerly Bro) [40], to help the SDN controller make decisions.

The current deployment requires some non-trivial pre-configuration for the PC boxes in shell script. We seek to ease this process with an automatic provisioning tool like Ansible [41] to configure new and existing microgrids. Since we already have the separation in SDN between the control plane and the application layer, we seek to provide a unified application layer to interface with both the SDN controller and Ansible.

7.2 Reflections

We have tried a variety of commercial hardware switches that support OpenFlow, such as HP Aruba 2920 and Pica8 P-3297. Unfortunately, none proved viable. Following are some of the issues we encountered: (a) lack of tunneling support or limitation of the number of remote tunneling end points; (b) enforcement of strict header matching where, for example, one must explicitly define both the source and destination addresses in the MAC header, thus blowing up the flow tables with unnecessary entries to an unwieldy proportion; (c) lack of support for optional protocols. The latter two issues constitute most of the impediment, because the OpenFlow standard leaves much room for manufacturers to decide which features they want to incorporate into their products, meaning two switches that support the same OpenFlow version could have different sets of functionalities, which, in turn, means that an SDN controller that works for switch A does not

necessarily work for switch B [42]. This significantly restricts interoperability and portability [42], [43].

Aside from the hardware restrictions, we believe there are flaws intrinsic to the OpenFlow model itself. Since version 1.0, it has gone through four major revisions [18]. Some of the key changes are as follows: version 1.1 added the support for MPLS and VLAN headers and introduced the concept of multi-layered flow tables; version 1.2 added the support for IPv6 and enhanced the support for existing protocols with additional fields; version 1.3 introduced flow metering and provided some interaction with tunneling interfaces with the "Tunnel-ID" metadata; version 1.4 enhanced the support for existing protocols with even more fields; the latest revision 1.5 introduced the concept of egress tables, allowing packets to be processed one more time after they are forwarded. One can see that a trend is to add more features for new protocols and improve support for existing ones. It is safe to speculate that VXLAN, what we use right now, might be included in the future. However, this model does not provide universality, especially with regard to uncommon or new protocol standards.

The Routing Service plays an important role in enabling DDS communication in conventional networks across the WAN under varying conditions of connectivity without the need to modify existing DDS applications. Due to the disparate characteristics in connectivity between LANs and WANs, the RS uses its own QoS profile for message filtering and delivery, mostly for conserving bandwidth and reducing loss rate. If we look at a packet from the sample capture of the RTPS in Appendix A, we can see that DDS provides rich data fields, such as Type Name, Reliability, Acknowledge Kind, Queue Size and Time-Based Filter. If we can take advantage of such fields, we can incorporate the QoS and IDS into the data plane and provide arbitrary filtering for DDS packets. This is harder for OpenFlow because it sought to commoditize hardware switches with fixed-function data planes, i.e., ASICs, most of which only recognize well-known protocols like Ethernet frame, IEEE 802.1Q, IPv4 or IPv6.

P4, a domain-specific programming language [44], however, seems to be a promising candidate to address these issues. With P4, we are no longer limited to software-defining only the control plane. We can also programmatically define the functionality of the data plane that can be either software (like OVS' kernel module) or hardware (like an FPGA that is able to pipeline packet processing at line rate). In any case, a programmable data plane allows the programmer to

freely define how to match a packet's header and what actions to take on it, coping with the ever-evolving networking field.

7.3 Conclusion

The cyber-physical system for the power grid is quickly evolving toward a smart-grid design with microgrids. We need a dynamic networking infrastructure to better suit the need of smart-grid applications. We believe that, with SDN, this is possible. We propose this SD-WAN approach to facilitate the communication for the OpenFMB-DDS framework and provide basic security measures. We show that individual devices can be isolated to protect the system against malicious applications with hardware switches, controllers and field device simulators. We also discuss other possible designs that trade efficiency for simplicity or use a hardware combination that seeks to strike a balance between expandability and performance. Lastly, we look at what additional security and protections we can provide for both power equipment and the networking infrastructure within the current framework. We also review some issues encountered along the way and reflect on the OpenFlow standard and the fundamental idea of SDN, thus providing insight into how we can achieve greater flexibility and the exact functionality we want.

References

[1] B. Lasseter, "Microgrids [distributed power generation]," in *2001 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.01CH37194)*, 2001.

[2] D. T. Ton and M. A. Smith, "The U.S. Department of Energy's Microgrid Initiative," *The Electricity Journal,* vol. 25, pp. 84-94, 2012.

[3] Modbus-IDA, "Modbus Application Protocol Specification v1.1a," 2004. [Online]. Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1a.pdf.

[4] R. E. Mackiewicz, "Overview of IEC 61850 and Benefits," in *2006 IEEE PES Power Systems Conference and Exposition*, 2006.

[5] K. Curtis, "A DNP3 protocol primer," *DNP User Group,* vol. 2005, 2005.

[6] E. Mallett, "NAESB Developing Framework for Current and Future Grid Interoperability," *Natural Gas & Electricity,* vol. 32, pp. 6-10, 2015.

[7] D. U. Case, "Analysis of the cyber attack on the Ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC),* 2016.

[8] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials,* vol. 16, pp. 1617-1634, 2014.

[9] M. Barnes, J. Kondoh, H. Asano, J. Oyarzabal, G. Ventakaramanan, R. Lasseter, N. Hatziargyriou and T. Green, "Real-World MicroGrids-An Overview," in *2007 IEEE International Conference on System of Systems Engineering*, 2007.

[10] A. Foster, *Messaging Technologies for the Industrial Internet and the Internet of Things Whitepaper. A comparison between DDS, AMQP, MQTT, JMS, REST and CoAP*, Version, 2014.

[11] Real-Time Innovations, "An Introduction to Connext DDS," 2015. [Online]. Available: https://community.rti.com/static/documentation/connext-dds/5.2.0/doc/manuals/connext_dds/html_files/RTI_ConnextDDS_CoreLibraries_GettingStarted/Content/GettingStarted/An_Introduction_to_.htm. [Accessed December 2018].

[12] Object Management Group, "About the Data Distribution Service Specification Version 1.4," 2015. [Online]. Available: https://www.omg.org/spec/DDS/About-DDS/.

[13] S. A. Thomas, in *IP Switching and Routing Essentials: Understanding RIP, OSPF, BGP, MPLS, CR-LDP, and RSVP-TE*, New York, Wiley, 2002.

[14] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014.

[15] E. S. Pilli, R. C. Joshi and R. Niyogi, "Network forensic frameworks: Survey and research challenges," *Digital Investigation,* vol. 7, pp. 14-27, 2010.

[16] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, M. Casado and K. Amidon, "The Design and Implementation of Open vSwitch.," in *NSDI*, 2015.

[17] Ryu Project Team, "Ryu SDN Framework," 2017. [Online]. Available: https://osrg.github.io/ryu/.

[18] "OpenFlow Switch Specification Version 1.5.1," Open Networking Foundation, 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[19] J. Zhang and C. A. Gunter, "Application-aware secure multicast for power grid communications," in *2010 First IEEE International Conference on Smart Grid Communications*, 2010.

[20] A. Adams, J. Nicholas and W. Siadak, "Protocol independent multicast-dense mode (PIM-DM): Protocol specification (revised)," 2004.

[21] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C.-G. Liu, P. Sharma and L. Wei, "Protocol independent multicast-sparse mode (PIM-SM): Protocol specification," 1998.

[22] W. Fenner, "Internet group management protocol, version 2," 1997.

[23] H. Holbrook, B. Cain and B. Haberman, "Using internet group management protocol version 3 (IGMPv3) and multicast listener discovery protocol version 2 (MLDv2) for source-specific multicast," 2006.

[24] T. Pfeiffenberger, J. L. Du, P. B. Arruda and A. Anzaloni, "Reliable and flexible communications for power systems: Fault-tolerant multicast with sdn/openflow," in *New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on*, 2015.

[25] A. Cahn, J. Hoyos, M. Hulse and E. Keller, "Software-defined energy communication networks: From substation automation to future smart grids," in *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, 2013.

[26] T. Humernbrum, B. Hagedorn and S. Gorlatch, "Towards efficient multicast communication in software-defined networks," in *Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on*, 2016.

[27] A. Goodney, S. Kumar, A. Ravi and Y. H. Cho, "Efficient PMU networking with software defined networks," in *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, 2013.

[28] Real-Time Innovations, "RTI Routing Service User's Manual," June 2017. [Online]. Available: https://community.rti.com/static/documentation/connext-dds/5.3.0/doc/manuals/routing_service/RTI_Routing_Service_UsersManual.pdf.

[29] S. Gordeychik, D. Kolegov and A. Nikolaev, "SD-WAN Internet Census," *arXiv preprint arXiv:1808.09027*, 2018.

[30] S. Chia, M. Gasparroni and P. Brick, "The next challenge for cellular networks: backhaul," *IEEE Microwave Magazine*, vol. 10, pp. 54-66, 8 2009.

[31] D. L. Blair, M. L. Sullenberger, S. T. Lucas, S. W. Wood and A. Oswal, *Intelligent wide area network (IWAN)*, Google Patents, 2017.

[32] T. Balan, D. Robu and F. Sandu, "LISP Optimisation of Mobile Data Streaming in Connected Societies," *Mobile Information Systems*, vol. 2016, 2016.

[33] D. Plummer, "Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48. bit Ethernet address for transmission on Ethernet hardware," 1982.

[34] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell and C. Wright, "Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks," 2014.

[35] J. Morales, A. Jimenez, A. Soto, I. Blancas and R. Wahlin, *RTI Connext DDS Performance Test,* GitHub.

[36] A. Odermatt, "Simplified OpenFMB DDS Based Demonstration," [Online]. Available: https://github.com/openfmb/openfmb-dds-demo.

[37] F. Clavel, E. Savary, P. Angays and A. Vieux-Melchior, "A network simulator for IEC61850 architecture," in *Petroleum and Chemical Industry Committee Conference*, Istanbul, 2013.

[38] S. Frankel and S. Krishnan, "IP security (IPsec) and internet key exchange (IKE) document roadmap," 2011.

[39] S. Smalley, "Configuring the SELinux policy," *NAI Labs Rep*, pp. 2-7, 2002.

[40] Paxson, Vern et al., "The Zeek network security monitor," [Online]. Available: https://www.zeek.org/.

[41] L. Hochstein and R. Moser, in *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*, O'Reilly Media, 2017.

[42] M. Kuzniar, P. Peresini, M. Canini, D. Venzano and D. Kostic, "A SOFT Way for Openflow Switch Interoperability Testing," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, New York, NY, USA, 2012.

[43] S. J. Vaughan-Nichols, "OpenFlow: The Next Generation of the Network?" *Computer*, vol. 44, pp. 13-15, 8 2011.

[44] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 87-95, 2014.

Appendix A

Sample Capture of an RTPS Packet

```
Frame 120: 530 bytes on wire (4240 bits), 530 bytes captured (4240 bits) on interface 0
Ethernet II, Src: Raspberr_33:a1:77 (b8:27:eb:33:a1:77), Dst: IPv4mcast_7f:00:01 (01:00:5e:7f:00:01)
Internet Protocol Version 4, Src: 172.16.1.2, Dst: 239.255.0.1
User Datagram Protocol, Src Port: 38147, Dst Port: 7400
Real-Time Publish-Subscribe Wire Protocol
   Magic: RTPS
   Protocol version: 2.1
      major: 2
      minor: 1
   vendorId: 01.01 (Real-Time Innovations, Inc. - Connext DDS)
   guidPrefix: ac100102000029aa00000001
      hostId: 0xac100102
      appId: 0x000029aa
      instanceId: 0x00000001
   Default port mapping: MULTICAST_METATRAFFIC, domainId=0
      [domain_id: 0]
      [traffic_nature: MULTICAST_METATRAFFIC (2)]
   submessageId: INFO_TS (0x09)
      Flags: 0x01, Endianness bit
         0... .... = Reserved: Not set
         .0.. .... = Reserved: Not set
         ..0. .... = Reserved: Not set
         ...0 .... = Reserved: Not set
         .... 0... = Reserved: Not set
         .... .0.. = Reserved: Not set
         .... ..0. = Timestamp flag: Not set
         .... ...1 = Endianness bit: Set
      octetsToNextHeader: 8
      Timestamp: Nov  2, 2018 19:53:29.000735999 UTC
   submessageId: DATA (0x15)
      Flags: 0x05, Data present, Endianness bit
         0... .... = Reserved: Not set
         .0.. .... = Reserved: Not set
         ..0. .... = Reserved: Not set
         ...0 .... = Reserved: Not set
         .... 0... = Serialized Key: Not set
         .... .1.. = Data present: Set
         .... ..0. = Inline QoS: Not set
         .... ...1 = Endianness bit: Set
      octetsToNextHeader: 3204
      0000 0000 0000 0000 = Extra flags: 0x0000
      Octets to inline QoS: 16
      readerEntityId: ENTITYID_UNKNOWN (0x00000000)
         readerEntityKey: 0x000000
         readerEntityKind: Application-defined unknown kind (0x00)
      writerEntityId: ENTITYID_BUILTIN_SUBSCRIPTIONS_WRITER (0x000004c2)
         writerEntityKey: 0x000004
         writerEntityKind: Built-in writer (with key) (0xc2)
      writerSeqNumber: 6
      serializedData
         encapsulation kind: PL_CDR_LE (0x0003)
         encapsulation options: 0x0000
         serializedData:
            PID_ENDPOINT_GUID
               parameterId: PID_ENDPOINT_GUID (0x005a)
               parameterLength: 16
```

Endpoint GUID: ac100102 000029aa 00000001 80000507
PID_TOPIC_NAME
parameterId: PID_TOPIC_NAME (0x0005)
parameterLength: 24
topic: BatteryEventProfile
PID_TYPE_NAME
parameterId: PID_TYPE_NAME (0x0007)
parameterLength: 68
typeName: OpenFMB_Information_Model::openfmb::essmodule::ESSEventProfile
PID_RELIABILITY
parameterId: PID_RELIABILITY (0x001a)
parameterLength: 12
Kind: RELIABLE_RELIABILITY_QOS (0x00000002)
PID_ACK_KIND
parameterId: PID_ACK_KIND (0x800b)
parameterLength: 4
Acknowledgment Kind: PROTOCOL_ACKNOWLEDGMENT (0x00000000)
PID_RECV_QUEUE_SIZE [deprecated]
parameterId: PID_RECV_QUEUE_SIZE [deprecated] (0x0018)
parameterLength: 4
queueSize: 0xffffffff
PID_TIME_BASED_FILTER
parameterId: PID_TIME_BASED_FILTER (0x0004)
parameterLength: 8
lease_duration: 0 sec
PID_LIVELINESS
parameterId: PID_LIVELINESS (0x001b)
parameterLength: 12
PID_LIVELINESS
PID_DURABILITY
parameterId: PID_DURABILITY (0x001d)
parameterLength: 4
Durability: VOLATILE_DURABILITY_QOS (0x00000000)
PID_DIRECT_COMMUNICATION
parameterId: PID_DIRECT_COMMUNICATION (0x8011)
parameterLength: 4
Direct Communication: True
PID_OWNERSHIP
parameterId: PID_OWNERSHIP (0x001f)
parameterLength: 4
Kind: SHARED_OWNERSHIP_QOS (0x00000000)
PID_PRESENTATION
parameterId: PID_PRESENTATION (0x0021)
parameterLength: 8
Access Scope: INSTANCE_PRESENTATION_QOS (0x00000000)
Coherent Access: False
Ordered Access: False
PID_DESTINATION_ORDER
parameterId: PID_DESTINATION_ORDER (0x0025)
parameterLength: 4
Kind: BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS (0x00000000)
PID_DEADLINE
parameterId: PID_DEADLINE (0x0023)
parameterLength: 8
lease_duration: INFINITE
PID_LATENCY_BUDGET
parameterId: PID_LATENCY_BUDGET (0x0027)
parameterLength: 8
lease_duration: 0 sec
PID_GROUP_ENTITY_ID
parameterId: PID_GROUP_ENTITY_ID (0x0053)
parameterLength: 4

Group entity ID: 0x80000509 (unknown kind (09): 0x800005)
PID_ENTITY_VIRTUAL_GUID
parameterId: PID_ENTITY_VIRTUAL_GUID (0x8002)
parameterLength: 16
guidPrefix: ac100102000029aa00000001
virtualGUIDSuffix: 0x80000507 (Application-defined reader (with key): 0x800005)
PID_SERVICE_KIND
parameterId: PID_SERVICE_KIND (0x8003)
parameterLength: 4
serviceKind: NO_SERVICE_QOS (0x00000000)
PID_PROTOCOL_VERSION
parameterId: PID_PROTOCOL_VERSION (0x0015)
parameterLength: 4
Protocol version: 2.1
PID_VENDOR_ID
parameterId: PID_VENDOR_ID (0x0016)
parameterLength: 4
vendorId: 01.01 (Real-Time Innovations, Inc. - Connext DDS)
PID_PRODUCT_VERSION
parameterId: PID_PRODUCT_VERSION (0x8000)
parameterLength: 4
Product version: 5.3.0.0
PID_DISABLE_POSITIVE_ACKS
parameterId: PID_DISABLE_POSITIVE_ACKS (0x8005)
parameterLength: 4
disablePositiveAcks: False
PID_EXPECTS_VIRTUAL_HB
parameterId: PID_EXPECTS_VIRTUAL_HB (0x8009)
parameterLength: 4
expectsVirtualHB: False
PID_ENTITY_NAME
parameterId: PID_ENTITY_NAME (0x0062)
parameterLength: 28
entityName: Battery Event Reader
PID_TYPE_CONSISTENCY
parameterId: PID_TYPE_CONSISTENCY (0x0074)
parameterLength: 4
Type Consistency Kind: ALLOW_TYPE_COERCION (0x0001)
PID_ENDPOINT_PROPERTY_CHANGE_EPOCH
parameterId: PID_ENDPOINT_PROPERTY_CHANGE_EPOCH (0x8015)
parameterLength: 8
Endpoint Property Change Epoch: 1
PID_TYPE_OBJECT
parameterId: PID_TYPE_OBJECT (0x0072)
parameterLength: 2788
Type Object
PID_ENDPOINT_SECURITY_ATTRIBUTES
parameterId: PID_ENDPOINT_SECURITY_ATTRIBUTES (0x8018)
parameterLength: 4
Flags: 0x00000000
PID_SENTINEL
parameterId: PID_SENTINEL (0x0001)