

A Software-Defined Networking (SDN) Assisted Middleware Interconnecting
Supervisory Control and Data Acquisition (SCADA) Systems

by

Beibei Liu

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2018 by the
Graduate Supervisory Committee:

Junshan Zhang, Chair
Vijay Vittal
Sau Kwan

ARIZONA STATE UNIVERSITY

August 2018

ABSTRACT

The reliable operation of critical infrastructure systems is of significant importance to society. The power grid and the water distribution system are two critical infrastructure systems, each of which is facilitated by a cyber-based supervisory control and data acquisition (SCADA) system. Although critical infrastructure systems are interdependent with each other due to coupling (a power grid may be the electrical supply for a water distribution system), the corresponding SCADA systems operated independently and did not share information with each other. Modern critical infrastructure systems tend to cover a larger geographic area, indicating that a SCADA control station supervising a small area is far from meeting the demands.

In this thesis, the above-mentioned problem is addressed by building a middleware to facilitate reliable and flexible communications between two or more SCADA systems. Software Defined Networking (SDN), an emerging technology providing programmable networking, is introduced to assist the middleware. In traditional networks, network configurations required highly skilled personnel for configuring many network elements. However, SDN separates the control plane from the data plane, making network intelligence logically centralized, and leaving the forwarding switches with easy commands to follow. In this way, the underlying network infrastructures can be easily manipulated by programming, supporting the future dynamic network functions.

In this work, an SDN-assisted middleware is designed and implemented with open source platforms Open Network Operating System (ONOS) and Mininet, connecting the power grids emulator and water delivery and treatment system (WDTS) emulator EPANet. Since the focus of this work is on facilitating communications between dedicated networks, data transmissions in backbone networks are emulated. For the interfaces, a multithreaded communication module is developed. It not only enables

real-time information exchange between two SCADA control centers but also supports multiple-to-multiple communications simultaneously. Human intervention is allowed in case of emergency.

SDN has many attractive benefits, however, there are still obstacles like high upgrade costs when implementing this technique. Therefore, rather than replace all the routers at once, incremental deployment of hybrid SDN networks consisting of both legacy routers and programmable SDN switches is adopted in this work. We emulate on the ratio of SDN deployment against the performance of the middleware and the results on the real dataset show that a higher fraction of SDN results in a higher reliability and flexibility of data transmissions. The middleware developed may contribute to the development of the next-generation SCADA systems.

To My Family.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Dr. Junshan Zhang for the continuous support of my master's studies and related research. His guidance helped me in shaping this research work. Special thanks to his enthusiasm, extraordinary patience and serious attitude towards research, which proved to be an immense help to me, all the time. I feel so honored and grateful to have been one of his students. Besides my advisor, I am grateful to Dr. Vijay Vittal and Dr. Sau Kwan for their precious time in serving on my thesis committee member and for their insightful comments and valuable feedback. I would like to thank Dr. Vijay Vittal, Dr. Larry Mays, Dr. Sau Kwan, Scott Zuloaga, and Puneet Nandkumar for their advice and discussions weekly over two and a half years. Without their precious support, it would be impossible to conduct this research. I would like to extend my appreciation to the School of Electrical, Computer and Energy Engineering at Arizona State University for providing me this opportunity to pursue my master's degree.

I would like to thank all my current and former colleagues in the Information, Energy and Networking Lab (IENL), Jun Zhao, Zhengyu Zhang, Weina Wang, Shaobo Zhang, Mehmet Dedeoglu, Yangjun Li, Sen Lin, Shi Yan, Abdullah Basar Akbay, Theo Chiu, Umut Demirhan, Zuyuan Zhang, Zhaofeng Zhang, Yaoming Zhuang, Anirban Bhattacharya, JK Jensen for their kindness, help and support. I would like to thank all my friends, Jingchao Sun, Sai Zhang, Jie Fan and Kausic Gunasekar for their generous help in my research and courses.

Most importantly, I would like to thank my parents, for their unconditional love and support, without whom, I could not have completed this work.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	vii
CHAPTER	
1 INTRODUCTION	1
2 BACKGROUND	5
2.1 Evolution of SCADA Systems	5
2.2 Introduction of SDN	6
2.2.1 Data Forwarding Plane	7
2.2.2 Control Plane	8
2.3 state-of-the-art and applications	9
2.3.1 State-of-the-Art	9
2.3.2 Applications	10
2.4 Summary	11
3 DESIGN OF MIDDLEWARE	12
3.1 Feedback with Controller in the Loop	12
3.2 Hybrid Deployment	13
3.3 SDN Island	13
3.4 Seamless Internetworking of SDN and IP	15
3.5 Related Work	16
3.6 Metrics of Performance	17
3.7 Virtual Layer	18
3.7.1 Metrics	18
3.7.2 Virtual Layer Building Algorithm	19
3.7.3 Stability Evaluation	19
3.8 Summary	20

CHAPTER	Page
4 IMPLEMENTATION OF MIDDLEWARE	23
4.1 Overall Function	23
4.2 Network Topology Data	24
4.3 System Setting	26
4.4 Program Architecture	30
4.5 Communication between Emulators and Middleware.....	35
4.5.1 Multi-threaded Communication	35
4.5.2 Relation to Emulator	37
4.5.3 Cross Platform Features	38
4.6 Apply the Module to the Platform	39
4.7 Varied Ratio of SDN Deployment	39
4.8 Performance Evaluations	39
4.9 Summary	42
5 CONCLUSION AND FUTURE WORK.....	43
5.1 Conclusion	43
5.2 Future Work	44
5.3 Summary	45
BIBLIOGRAPHY	46

LIST OF FIGURES

Figure	Page
1.1 the Communication System between Two SCADA Systems	3
2.1 the Involvement of SCADA Communication System	5
2.2 Framework of SDN.....	7
2.3 Architecture of ONOS	9
3.1 SDN Controller in a Feedback Loop[15]	13
3.2 SDN Islands in Middleware	15
4.1 Implementation of SDN assisted Middleware Connecting SCADA Sys- tems	24
4.2 Stanford Large Network Dataset Collection - SNAP	25
4.3 Network Topology Data Imported From Dataset	25
4.4 Network Topology with Hybrid Deployment.....	26
4.5 Mininet in VirtualBox	27
4.6 Open Networking Operating System (ONOS)	27
4.7 Emulator Topology	28
4.8 Different SDN Deployment Rate	29
4.9 IP Addresses of Hosts	30
4.10 Packet Flows in the Networks.....	31
4.11 All Flows Going through a Given SDN Switch	32
4.12 Intent Conveys Policies to Data Plane	33
4.13 Components for SDN-IP.....	34
4.14 Diagrams for the Program	35
4.15 Multithreaded Communication Module	36
4.16 Block Diagram of Communication Module	38
4.17 Install and Activate the Applications in ONOS.....	40

Figure	Page
4.18 Varied Ratio of SDN Deployment	40
4.19 Experiment Results:End-to-End Delay.....	41
4.20 Experiment Results:Throughput	42

Chapter 1

INTRODUCTION

Supervisory Control and Data Acquisition (SCADA) systems are a class of control system architecture utilizing computers, networked data communications, and graphical user interfaces for high-level process supervisory management [21]. It is widely used in many critical infrastructures, such as power grids, to collect real-time data from sensors, analyze operational status, and provide instructions to the actuators. SCADA systems are of vital importance for the reliable operation of critical infrastructures, especially in severe conditions like mega-droughts or during an unexpected attack.

Power grids and water distribution system are two examples of critical infrastructures, which are sometimes physically interdependent. For example, power grids provide electricity for water distribution systems, while water systems provide the power plants with cooling water. However, the respective SCADA systems historically have operated independently without sharing information. To enable communications between two or more SCADA systems, an SDN-assisted middleware is designed in this thesis.

Previously, SCADA systems utilized proprietary equipment and dedicated networks. Today, since SCADA systems expand over larger geographical areas, they incorporate Local Area Networking (LAN), Wide Area Networking (WAN) or the Internet [16] to communicate within the system or collaborate with other SCADA systems. Therefore, in this work, the backbone of the networking is considered.

An autonomous system (AS) is considered to be a region of networks belonging to a shared entity or running the same protocol. As of early 2018, the Internet consisted of

over 60,000 AS entities [19]. Two or more SCADA systems are likely located within a different AS that are both connected to the backbone network, which typically adopts the Border Gateway Protocol (BGP). When the network topology changes or traffic shifts drastically, a BGP update is triggered, which consumes significant CPU resources and possibly causes slower convergence, thereby inducing heavy packet losses. BGP supports some complicated network policies from different entities, which adds substantially to the complexity [3], and one solution is to adopt a new routing mechanism. In this work, Software Defined Networking (SDN) is introduced, which overcomes many disadvantages of the traditional decentralized network, such as complexity and the heavy dependence on connectivity structure [25, 26, 26, 28]. It separates the control plane from the data plane and makes the routing more reliable and flexible.

In this work, we design and implement an SDN-assisted middleware for interconnecting power grids and water distribution systems, as illustrated in figure 1.1. The middleware is a virtual layer built between the application layer and the underlying routing layer. The controller in the application layer determines the packet forwarding behaviors. The topology and traffic information controller requirements are collected through the middleware.

Although SDN outperforms the general routing scheme, it remains unrealistic to upgrade all routers simultaneously due to significant costs. Instead, it is reasonable to assume incremental deployment through deploying SDN on the selected routers and realizing the customized routing between individual SCADA systems. We analyze this hybrid deployment model for SDN switches and legacy routers.

Finally, the proposed middleware is implemented on the open source platform, Open Networking Operating System (ONOS), Mininet, and the routing software Quagga. The results demonstrate that the customized routing, with the aid of SDN, out-

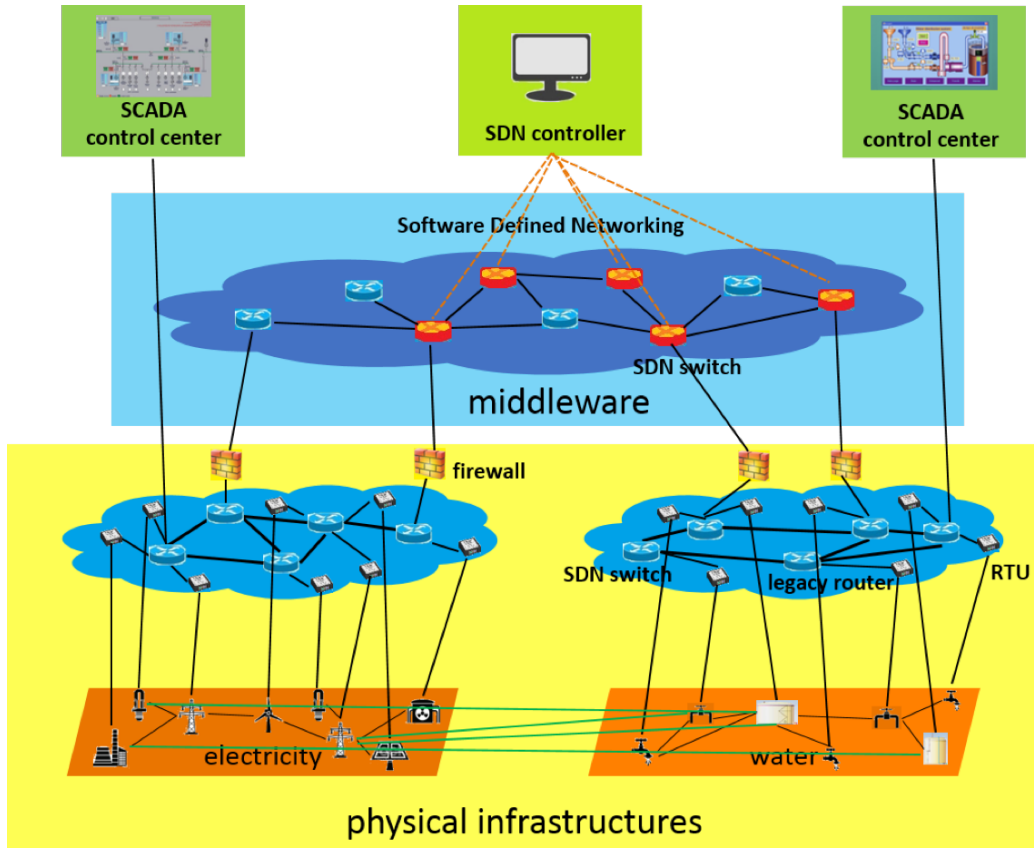


Figure 1.1: the Communication System between Two SCADA Systems

performs the general routing in terms of flexibility and reliability, which are critical factors for SCADA systems. Regarding interfaces, we develop a module enabling real-time communication between two or more SCADA systems, which allows bi-direction data transmissions.

The proposed middleware offers a wide range of applications. It can interconnect to SCADA systems as well as enterprise networks and data centers. Internet Service Providers (ISPs) can profit by selling this special virtual middleware service to utilities. It also provides insight into the design and implementation of next-generation SCADA systems [12].

The contributions of this work include:

1. Propose a middleware method to solve the cooperation problems between otherwise isolated SCADA systems, enabling efficient communications between SCADA systems.
2. Introduce an SDN technique offering flexible and reliable customized routing services. SDN provides possibilities of innovations towards next-generation networking.
3. Adopt a hybrid deployment of SDN switches and legacy routers to avoid the significant costs of upgrading all devices simultaneously, which is practical for implementations.
4. Implement the design using network emulators Mininet, software router Quagga, and the open source controller ONOS, paving the way for future hardware deployment.
5. Develop middleware not only for SCADA systems but also applicable to communications between enterprise networks and data centers.

The thesis follows with the background of SCADA systems and SDN in Chapter 2 as well as state of the art for this technology. Chapter 3 describes the middleware design, including the algorithms for formulating the deployment patterns. Chapter 4 outlines the implementation details of the middleware and demonstrates the results of the hybrid deployment. Chapter 5 concludes the work and identifies future work.

Chapter 2

BACKGROUND

2.1 Evolution of SCADA Systems

Figure 2.1 summarizes the evolution of SCADA systems through three stages. The first-generation monolithic SCADA systems were standalone with limited or connectivity to other systems. Networks were implemented exclusively for the control center to communicate with Remote Terminal Units (RTUs). The RTU equipment vendors often developed the communication protocols used on SCADA networks without accommodating other types of data traffic [4].

The second-generation distributed SCADA systems took advantage of developments

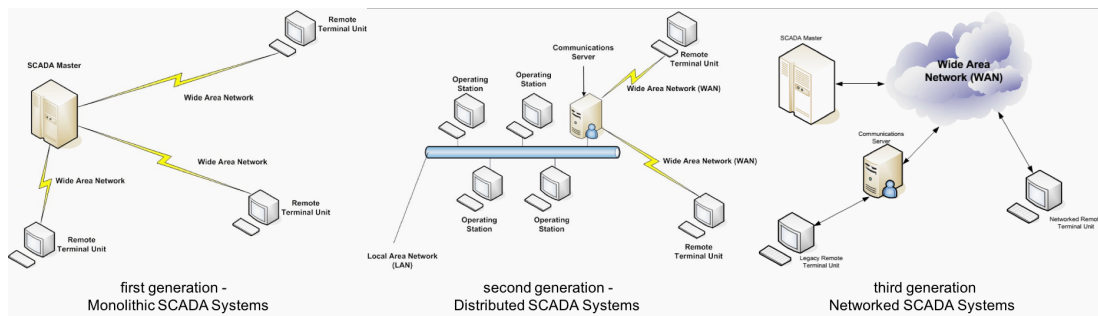


Figure 2.1: the Involvement of SCADA Communication System

in system and LAN technologies to distribute the processing across multiple stations, each with a specific function, such as historical data storage or real-time data processing. The networks connecting the individual stations and RTUs were based on LAN protocols and not capable of reaching beyond the local network.

The third-generation networked SCADA systems adopted an open system architecture by utilizing open standards and protocols, such as Internet Protocol (IP), instead

of proprietary vendor-controlled protocols. This approach enabled the distribution of SCADA functionality across a WAN without limitation to only a LAN. SCADA system engineers could then concentrate on the development of master stations and higher-level software.

2.2 Introduction of SDN

Software Defined Networking (SDN) is an emerging technique in networking. Unlike the practice in traditional networks where each router decides its forwarding behaviors based on a pre-defined routing protocol, SDN leverages the intelligence from individual routers and centralizes the control to a controller. This allows the network operations to be directly programmable, and the underlying infrastructure to be abstracted and easily manipulated by applications.

The SDN controller dynamically interprets the intention of applications, and injects rules into SDN-capable switches. According to these rules, the SDN switches perform packet inspection, manipulation, and forwarding. They can inspect and modify packet headers at different levels of the protocol stack from the data link layer to application layer. The framework of SDN is illustrated in figure `reffig:sdn-framework`.

The goal of SDN is to support the rapid response of network administrators to changing topology, traffic or business requirements via a centralized controller. In our work, we deploy SDN in the backbone networks transmitting packet flows to facilitate reliable and flexible data exchange between SCADA systems. With the aid of SDN, adapting to situations is easier, and a low packet loss rate and flexible packet forwarding can be achieved.

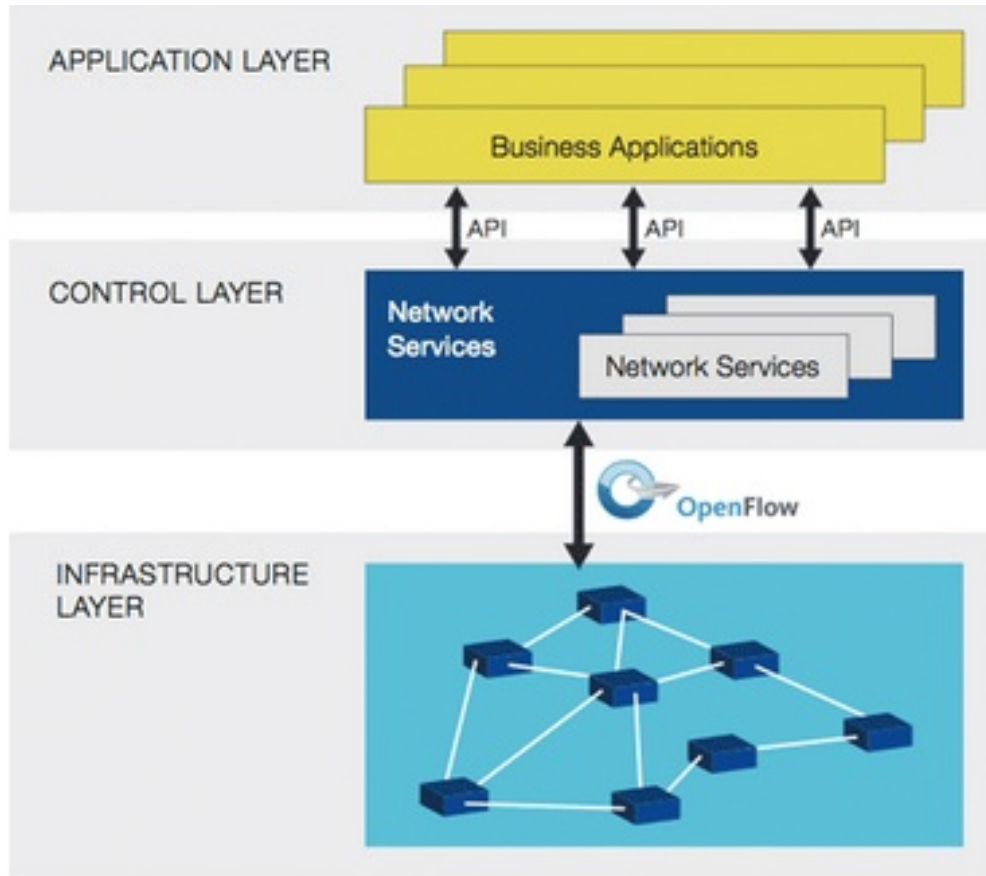


Figure 2.2: Framework of SDN

2.2.1 Data Forwarding Plane

The most salient feature of SDN is that it separates the control plane from data forwarding plane. Instead of allowing legacy routers to make forwarding decisions, SDN only requires switches to process simple commands, such as "match-action." Controllers generate these commands and transmit directly to SDN switches. Intelligence is centralized, which is one reason to call these components a "switch" instead of a "router."

Mininet is a network emulator developed for rapidly prototyping large networks by creating a virtual network running a real kernel on a single machine. This is realized

by using different namespaces, which creates an illusion that each node is running on a separate machine. Mininet provides a straightforward and extensible Python Application Programmable Interface (API) for network creation and emulations. Mininet networks run real code, including standard Unix/Linux network applications, as well as the real Linux kernel and network stack. The code developed and tested on Mininet for controllers, switches or hosts can move to a real system with minimal changes to support real-world testing, performance evaluation, and production deployment. This feature enables the development of designs that if they work in Mininet, then it is expected they can be easily moved to hardware switches for packet forwarding.

2.2.2 Control Plane

Various SDN controllers exist in the market, including ONOS [2], which stands out because it is a newly released open-source SDN controller focusing on service provider use cases. Therefore, it is considered to be the best candidate for backbone networks as it satisfies our needs as we are interested in backbone networks, which are operated by service providers. The architecture of ONOS is illustrated in figure 2.3. It is a composition of different layers. Northbound and Southbound are abstracted interfaces toward the application layer and the packet forwarding layer, respectively. In this work, we focus on Northbound by defining the controlling behaviors.

The ONOS platform is developed in Java as a multi-module project with modules managed as OSGi (Open Service Gateway Initiative) bundles in the Karaf environment. An advantage of using OSGi is that applications can be developed independently and deployed to the platform. In this work, the customized modules are developed based on the ONOS platform, and then loaded to the platform and activated, which is easier and safer.

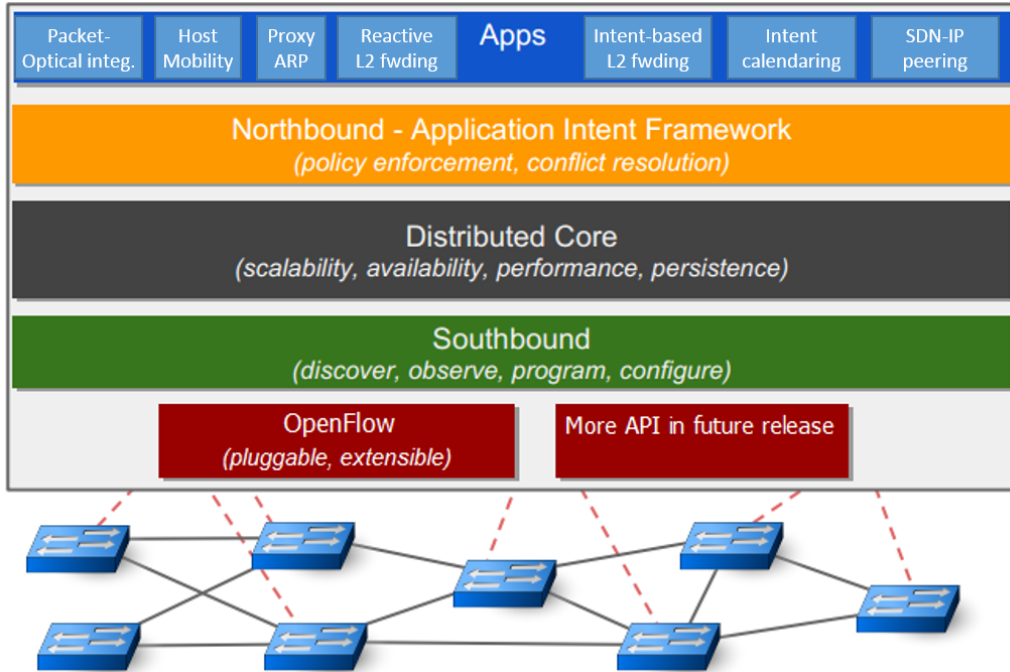


Figure 2.3: Architecture of ONOS

2.3 state-of-the-art and applications

2.3.1 State-of-the-Art

SDN significantly influences both academic and industrial research. Today, OpenFlow is a standardized protocol between the controller and switches, and switch forwarding tables can be populated directly via OpenFlow. Many commercial switches, including HP, NEC, and Pronto [6], support the OpenFlow APIs. Before the emergence of OpenFlow, the ideas for a programmable network were proposed for decades. From the mid-1990s to early 2000s, active networks introduced programmable functions into the network. Around 2001 to 2007, control and data planes were separated, and from 2007 to 2010, the OpenFlow API and network operating systems were developed [6].

Programmability in the network lowered the barrier to innovations. Recently, Google

B4 networks [8] adopted the prevailing Software-Defined Wide Area Networking (SD-WAN), and many other academic and industrial groups are participating in the field of SDN.

2.3.2 Applications

In this work, the communication between SCADA systems is discussed, although applications of SDN-assisted middleware extend to enterprise networks and data centers. Additional scenarios where SDN or SDN-assisted middleware can be valuable range from network engineering, the Internet of Things (IoT), and mobile communications.

SDN dramatically enhances the quality of network traffic engineering as it offers a unified and global view of complex networks providing a powerful control environment for network flows [1]. SDN-assisted middleware serves as a good platform IoTs. A comprehensive software-defined based framework model simplifies the IoT management process and provides a vital solution for the challenges in the traditional IoT architectures [9]. SDN-assisted middleware can be used in mobile communications to increase the efficiency, scalability, and elasticity of the Wireless Sensor Networks (WSNs) by estimating functions of initial measurements [23, 29] or learning global data information at sensors [24, 27].

SDN introduces the concept of Software-Defined, which expands opportunities for automation and innovation. We believe there will be more systems adopting these types of programmable features in the near future.

2.4 Summary

In this chapter, we introduced the evolution of SCADA systems and the background of SDN. The irresistible trend of enlarging SCADA systems expedites the development of communication systems to evolve from small, dedicated networks to large, public data networks. SDN brings about a new opportunity for SCADA communication systems by enabling the programmability of networking and shielding the difference in hardware equipment, policies or protocols. These advancements are of great importance to next-generation networking technologies.

Chapter 3

DESIGN OF MIDDLEWARE

3.1 Feedback with Controller in the Loop

Despite the rigid mechanism of running predefined protocols in the routers, SDN offers a new perspective for controlling the networking.

By modifying the header of packets, the SDN controller can block, redirect, or split the traffic to diverse neighborhoods. The controller manipulates the forwarding logic centrally, to affect the packet forwarding behaviors in the data plane. The actual traffic changes in the data plane will in return provide references for the controllers decision process. This formulates a feedback loop with the controller in the loop, illustrated in figure 3.1.

The upper cloud in figure 3.1 represents the application layer, like the network controlling authorities or Internet Service Providers (ISPs). The lower cloud stands for network resources, the links and intermediate nodes for data transmissions. The controller receives both the optimization policies and service requests from the application layer and the actual network resource states (bandwidth, delay, etc.) from the lower forwarding layer. Then, it determines the rules for forwarding a given flow of packets and modifying the resource state by issuing match-action commands to the forwarding plane.

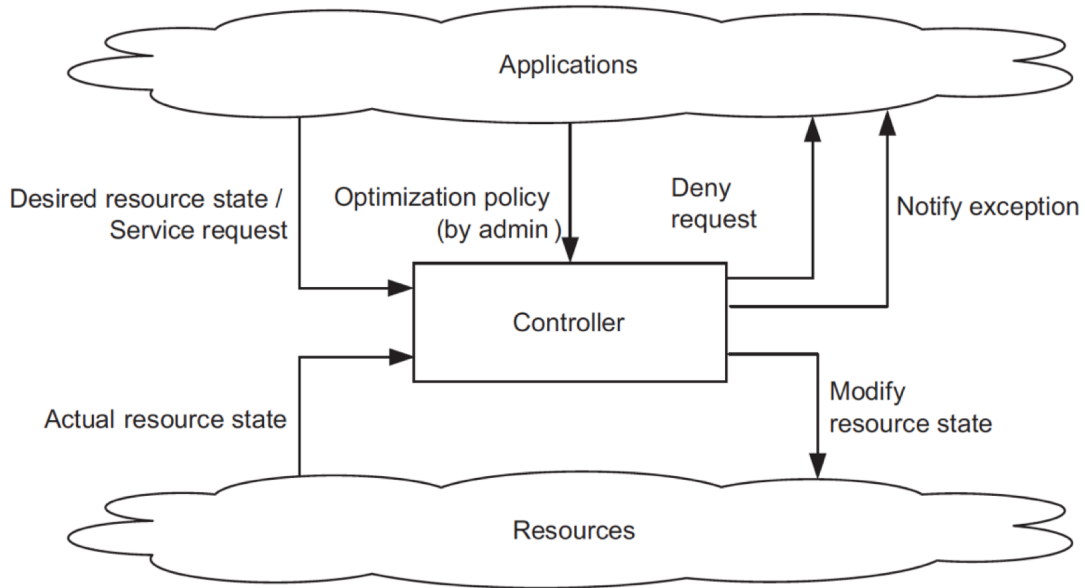


Figure 3.1: SDN Controller in a Feedback Loop[15]

3.2 Hybrid Deployment

To avoid replacing all the routers at one time, we adopt a hybrid deployment of SDN switches and legacy routers. Beginning with the partial deployment of SDN, we pick some important nodes to deploy SDN, under a limited budget. Then, as the better performances of SDN-assisted network can be seen by ISPs and network authorities, more SDN deployment will be gradually introduced.

3.3 SDN Island

SDN islands are subsections of networks that are SDN controlled. Each island may have one or more hops through a legacy network in between it and another island [20].

The SDN controller provides a global view of the network. However, in the hybrid

deployment setting, Link Layer Discovery Protocol (LLDP) probes are dropped at the edge of the SDN-controlled network and legacy network. This may give the illusion to the controller that those are separate networks. Open Networking Operating System (ONOS) uses Broadcast Domain Discovery Protocol (BDDP) probes to discover paths between islands, as legacy devices do not drop or mark BDDPs. Connections across legacy networks will appear as single hops regardless of how many legacy devices may be along that path [20].

Therefore, the less fragmented the network topology is, the better global view the controller will get. Otherwise, the topology that the controller sees is different from the real topology. To compensate it, a penalty is added to the boundary. SDN-IP is a module on the boundary to let SDN islands communicate through Border Gateway Protocol (BGP) at the data-plane level.

The middleware with SDN island views is illustrated in the figure 3.2. Legacy routers are grouped into a few SDN islands, represented by small clouds. The larger cloud above is the abstraction of SDN islands, seen from the controller perspective.

the whole communication network: $G = G_1 \cup G_2 \cup \dots \cup G_N$, where $G_k, k = 1, 2, \dots, N$ is an Autonomous System (AS).

$G_k = \{V, E\}$, V is the vertex set, E is the edge set.

$V = \{V_s, V_l\}$, where V_s is the subset of SDN nodes, V_l is the subset of legacy nodes.

$E = \{E_{ss}, E_{ll}, E_{sl}\}$, where E_{ss} is the subset of edges joining two SDN nodes, E_{ll} is the subset of edges joining two legacy nodes, E_{sl} is the subset of edges joining one SDN node and one legacy node.

Definition 3.3.1. *legacy region A legacy region G_l is a connected subgraph of the graph G , any node $v \in V[G_l]$ is legacy node. G_l contains all E_{ll} edges with both vertices in G_l and all E_{sl} edges with one of the vertices in G_l .*

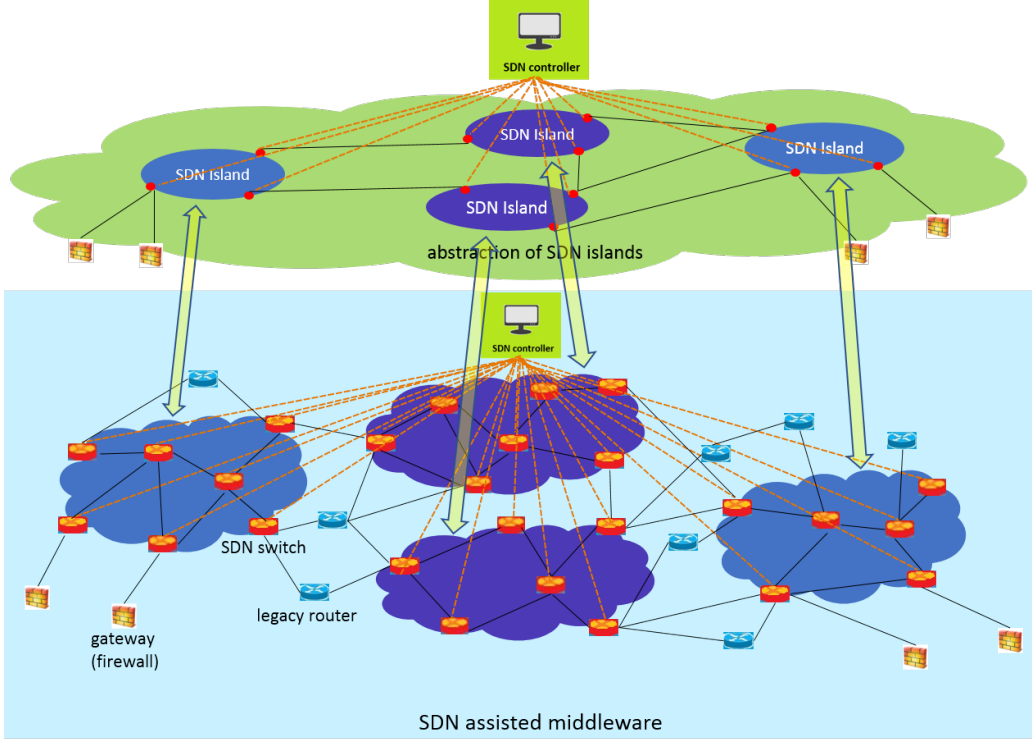


Figure 3.2: SDN Islands in Middleware

Definition 3.3.2. *frontier of legacy region* A frontier F^m of the m^{th} legacy region G_l^m is a subset of the edge set $E[G]$. F^m contains E_{sl} edges with one of the vertices in G_s , the other in G_l .

Definition 3.3.3. *path cost* path cost PC_p for a path $p = \{(p \cap E_{ss}) \cup (p \cap E_{ll}) \cup (p \cap E_{sl})\}$ is the sum of the costs on every edge from the beginning node to the end node (without loop). $PC_p = \sum_i \beta \cdot (p \cap E_{ss}) + \delta \cdot (p \cap E_{ll}) + \gamma \cdot (p \cap E_{sl})$ where β, δ, γ are the average link cost of three kinds of links (defined in section 3), respectively.

3.4 Seamless Internetworking of SDN and IP

After collecting the topology information of the network, we can choose either a predetermined SDN deployment plan or a fixed ratio of deployment plan. The former characterizes different shapes of SDN islands (a collection of SDN switches). The

latter describes the level of SDN penetration. In either way, the concrete deployment plan is determined.

One challenge lies in the place where the SDN switches and legacy routers interact. Internal BGP Speaker sits inside our SDN network and its job is to peer with all the External BGP Routers (running on software router Quagga), learn BGP routes from them, and relay those routes to the SDN-IP application running on ONOS. Therefore, messages can be conveyed and understood by each other, thus hybrid deployment is realized. The module working on this process is called the SDN-IP module.

ONOS used to be confined in his wonderful world, using OpenFlow protocol, unable to communicate with the outside. SDN-IP came and gave the ability to ONOS to speak to the rest of the world. ONOS started to communicate with IP components similar to it, using a widely used protocol BGP, which is a traditional communication protocol.

3.5 Related Work

In the legacy networks, BGP is the most common protocol for interdomain communications. F. Wang [17] finds that while routing changes can lead to longer delay and out-of-order packets, routing changes impact end-to-end loss more significantly and can lead to loss bursts lasting as long as 20 seconds.

Prior to the introduction of SDN, Route Control Platform (RCP) had been developed to control interdomain routing centrally. RCP computes the BGP routing table for an entire AS in a centralized server, and distributes the BGP routing table to individual routers. However, the intradomain routing is still performed in individual routers.

V. Kotronis [11] notices that the control plane can take several minutes to converge after a routing change; this may be unacceptable for real-time network services. Mul-

tidomain SDN centralization can be achieved by outsourcing routing functions to an external contractor, so that interdomain routing benefits through a birds eye view over several networks. SDN centralization helps linearly to reduce BGP convergence times and churn rates with expanding SDN deployments.

D. Lin [14] proposes an interdomain routing brokerage framework, and showed that an inter-AS routing path can be totally dominated by a small set of ASs and IXPs to provide end-to-end QoS guarantee. Lin models the problem as a Maximum Coverage with B-dominating path Guarantee (MCBG) problem.

W. Wang [18] plans SDN placement to enhance the SDN controllability over the hybrid network. Wang defines *PathControllability* to make sure that controllability should remain even if flows leave SDN nodes to ensure the effectiveness of SDN control. However, in our work we define the controllability of a region.

W. Wang plans SDN placement to enhance the SDN controllability over the hybrid network. Wang defines *PathControllability* to make sure that controllability should remain even if flows leave SDN nodes to ensure the effectiveness of SDN control. However, in our work we define the controllability of a region.

For dynamically admitting new SDN switches, R. Katiyar [10] proposes an autoconfiguration mechanism to provide configuration parameters to the newly detected SDN switch, overcoming the difficulty of a heterogeneous hybrid network.

3.6 Metrics of Performance

The backbone networks are usually run by ISPs, conforming to carrier-grade standards. In carrier-grade networks, the SDN control plane has to support high throughput to meet carrier requirements. The throughput of the SDN control plane can be evaluated by measuring its throughput under load at the northbound and southbound

interfaces.

In addition, failures or fluctuations occur and the SDN control plane needs to be able to detect and react to these failures swiftly to ensure seamless operation. Reaction to fluctuated traffic or unexpected network problems can be approximated by measuring the latency of the SDN control plane. This is why we choose topology change latency, flow/intent operation throughput as key metrics for ONOS performance.

3.7 Virtual Layer

3.7.1 Metrics

Proposition 3.7.1. *distance between nodes distance between nodes $d_{A,B}$ is a value measuring the total costs if a packet travels from node A to node B. If two nodes are joint by a single link, then $d_{A,B}$ is the cost of that link. If not, then $d_{A,B}$ is the cost of minimum sum of link costs joining two nodes.*

Proposition 3.7.2. *proportion of overlap proportion of overlap $Prop\{p_a, p_b\}$ is a value measuring how much two paths are overlapped.*

$$Prop\{p_a, p_b\} = \sum_{i,j} I_{a_i, b_j} \cdot Cost(a_i)$$

for any i, j pairs, where I_{a_i, b_j} is an indicator, $I_{a_i, b_j} = 1$ if a_i and b_j are the same link.

Proposition 3.7.3. *path similarity Path similarity $PS\{p_a, p_b\}$ is a metric measuring how much impact on end-to-end performance if a packet changes its path from $p_a = \{a_1, a_2, \dots, a_m\}$ to $p_b = \{b_1, b_2, \dots, b_n\}$.*

1. *distance between starting nodes d_{A_1, B_1} + distance between ending nodes d_{A_m, B_n}*

2. *difference of the length + sum of costs*

3. *proportion of overlap $Prop\{p_a, p_b\}$*

3.7.2 Virtual Layer Building Algorithm

We designed an offlineonline combined algorithm to build the virtual layer (middleware) between the networking layer and the application layer. The offline part (see algorithm 1) of the algorithm is based on the topology. The online part (see algorithm 2) is based on both the topology changes and traffic changes.

3.7.3 Stability Evaluation

There are various ways of defining stability. The goal of our virtual layer is to suppress small fluctuations while keeping or even amplifying large fluctuations. Because the small fluctuations do not have significant effect on routing, the system suppresses them to avoid unnecessary routing table changes. However, if the fluctuation is large enough, the system must pay attention to it and make corresponding changes in the routing table to tackle the problem.

Proposition 3.7.4. *excitation excitation of the AS routing system $Excit$ comprises of two perspectives:*

1. *extent of topology change. $Topo_C$ is the difference between the influence on routing (packet traversing the given AS) from the previous topology and the influence from current topology. $Topo_C = \frac{Size(Virt_E[G])}{Size(E[G])}$, where $Size(Virt_E[G])$ is the number of virtual links that this specific changing link involves. $Size(E[G])$ is the number of links in the whole network.*

2. *drastic traffic change.* $Traff_C$ is the amount that an elephant traffic going through a specific link far exceed the average traffic. $Traff_C = \frac{T}{Expct[T]}$.

$$Excit = Topo_C + Traff_C$$

The stability we are talking about are within the range of $excitation \in [0, \omega]$

3.8 Summary

In this chapter, we introduced various design aspects when building an SDN-assisted middleware. Apart from the overall design idea that putting the controller in the feedback loop, we also introduced the concept of SDN island, which is a building block of the SDN-IP hybrid deployment scheme. Then the algorithms for formulating the virtual layer was elaborated in detail. For the performance evaluation, some metrics were introduced or defined.

Algorithm 1 virtual layer building algorithm (offline part)

Input: adjacency matrix Q , maximum AS delay Max_D , **Output:** virtual layer $Virt_G = \{Virt_G_1, Virt_G_2, \dots, Virt_G_s\}$, $s = m \cdot n$ where each element $Virt_G_i$, $i = 1, \dots, s$ is a graph with simplified topology, corresponding to a virtual sub-layer.

- 1: define $key = \{a_{11}, a_{12}, \dots, a_{21}, \dots, a_{mn}\}$: set of all elements of the transfer matrix $A^{m \times n}$ (defined in section 4.B). The size of key is $m \cdot n$. Each element key_i , $i = 1, \dots, m \cdot n$ represents an $\{ingress, egress\}$ pair.
 - 2: define $U_{key} = \{U_{key1}, U_{key2}, \dots, U_{keys}\}$, $s = m \cdot n$, where U_{keyi} , $i = 1, \dots, s$ is the temporary virtual path serving the key_i $\{ingress, egress\}$ pair.
 - 3: define $Group = \{g_1, g_2, \dots, g_s\}$, where each element is a set of valid paths for a given $\{ingress, egress\}$ pair.
 - 4: **for** i in m **do**
 - 5: **for** j in n **do**
 - 6: find the shortest path p^* from ingress point x_i to each egress points x_j , using Shortest Path First algorithm.
 - 7: **while** path similarity $PS\{p_i, p_* < threshold \ \varrho\}$ **do**
 - 8: $i++$, get another path.
 - 9: add path p_i to $g_{(i-1) \cdot j + j}$
 - 10: **for** i in m **do**
 - 11: **for** j in n **do**
 - 12: $a_{ij} = key_{(i-1) \cdot j + j}$
 - 13: assign each element in the $Group = \{g_1, g_2, \dots, g_s\}$ as an aggregated virtual path.
- return** $Group$
-

Algorithm 2 virtual layer building algorithm (online part)

Input: $Group = \{g_1, g_2, \dots, g_s\}$, topology change matrix $Topo_C$, traffic change matrix $Traff_C$, signal $I_{topo} = 1, I_{Traff} = 1$, indicating if there is topology/traffic changes, respectively, **Output:** Abstract virtual layer $AbsVirt_g = \{AbsVirt_g_1, AbsVirt_g_2, \dots, AbsVirt_g_s\}, s = m \cdot n$ where each element $AbsVirt_g_i, i = 1, \dots, s$ is a virtual layer solution.

- 1: import the output $Group$ from the off-line algorithm
 - 2: **if** $I_{topo} = 1$ **then**
 - 3: **while** non-zero element exists **do**
 - 4: find the coordinates of of non-zero elements, mapping to a link e_i
 - 5: **for** k in $s, s = m \cdot n$ **do**
 - 6: **if** $e_i \in g_k$ **then**
 - 7: **return** $AbsVirt_g$
-

Chapter 4

IMPLEMENTATION OF MIDDLEWARE

4.1 Overall Function

The goal of the SDN-assisted middleware is to connect two or more SCADA systems, which touch the real-world SCADA systems and all networks involved. In our emulation, SCADA systems are represented by sophisticated computer models such as the EPANET model, while the proposed middleware is represented by a Linux hosting machine (control plane) as well as the virtual-box (data plane) embedded in it. Communication modules are the interfaces between SCADA systems and the middleware. The correspondence between real system and emulators is illustrated in Figure 4.1.

In real SCADA systems, raw sensor data (e.g., water level information) are generated and put into a buffer of Remote Terminal Units (RTUs). Then, the RTU is responsible for reporting all data to the control center servers (including real-time data server, historical data server, etc.). This process can be done either through a wireless hotspot or through wired Local Area Networks (LANs). Raw data must go through a state generator before they make sense to the SCADA control centers. Afterward, data processed by a state generator will be shown on HumanMachine Interfaces (HMIs) or stored in servers.

In this work, computer models of power grids and water distribution systems are built through emulators. The raw data collection process is out of the scope of this work. We focus on the process that data are transmitted from one SCADA control center (server) to one or more other SCADA control centers (servers), traversing the

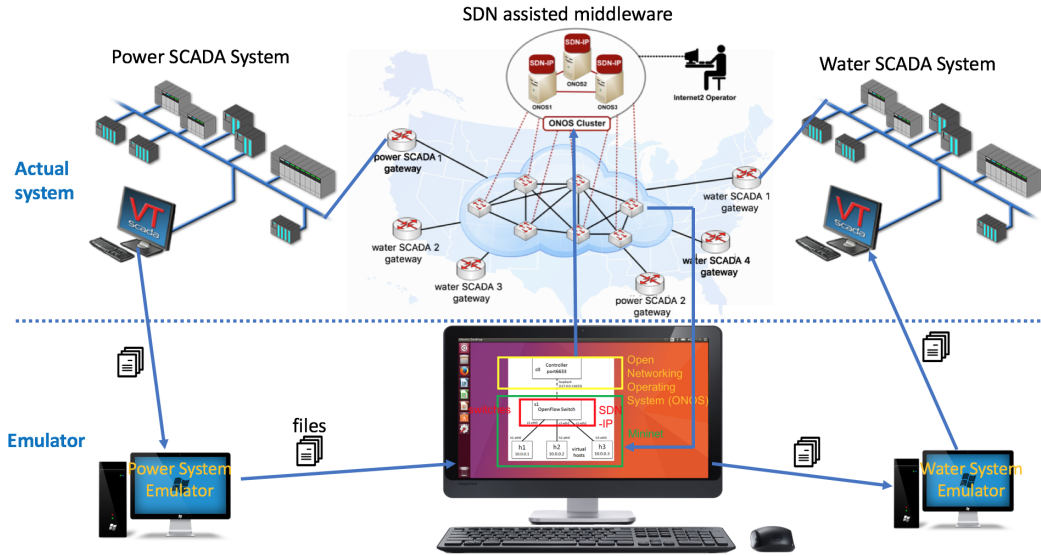


Figure 4.1: Implementation of SDN assisted Middleware Connecting SCADA Systems backbone networks, Wide Area Networks (WANs), or even the Internet.

4.2 Network Topology Data

We utilize the free data set from some enterprises or universities. For example, Amazons AWS Public Data Sets program and Stanfords Large Network Dataset Collection (SNAP) provide real-world networking data to the public for free. Figure 4.2 is a screenshot of the SNAP data set. The real network data outperform synthesized network data as they fit better to performance in real systems. Data are introduced to the network emulator Mininet with Python scripts. Figure 4.3 is a sketch of the imported topology, plotted with R packages. Although a real-world data set is more practical, we use a smaller data set here for the ease of a clear presentation. Visualization of the demo topology is shown in figure 4.4. Circles (blue) with bold lines are SDN switches, while others (red) are legacy routers. Links between SDN switch and



- SNAP for C++ >
- SNAP for Python >
- SNAP Datasets >
- What's new
- People
- Papers
- Projects >
- Citing SNAP
- Links
- About
- Contact us

Open positions

We have filled all the positions for this quarter. [More info.](#)

Stanford Large Network Dataset Collection

- **Social networks** : online social networks, edges represent interactions between people
- **Networks with ground-truth communities** : ground-truth network communities in social and information networks
- **Communication networks** : email communication networks with edges representing communication
- **Citation networks** : nodes represent papers, edges represent citations
- **Collaboration networks** : nodes represent scientists, edges represent collaborations (co-authoring a paper)
- **Web graphs** : nodes represent webpages and edges are hyperlinks
- **Amazon networks** : nodes represent products and edges link commonly co-purchased products
- **Internet networks** : nodes represent computers and edges communication
- **Road networks** : nodes represent intersections and edges roads connecting the intersections
- **Autonomous systems** : graphs of the internet
- **Signed networks** : networks with positive and negative edges (friend/foe, trust/distrust)
- **Location-based online social networks** : Social networks with geographic check-ins
- **Wikipedia networks, articles, and metadata** : Talk, editing, voting, and article data from Wikipedia
- **Temporal networks** : networks where edges have timestamps
- **Twitter and Memetracker** : Memetracker phrases, links and 467 million Tweets
- **Online communities** : Data from online communities such as Reddit and Flickr
- **Online reviews** : Data from online review systems such as BeerAdvocate and Amazon

SNAP networks are also available from [UF Sparse Matrix collection](#). [Visualizations of SNAP networks](#) by Tim Davis.

Social networks

Name	Type	Nodes	Edges	Description
ego-Facebook	Undirected	4,039	88,234	Social circles from Facebook (anonymized)
ego-Gplus	Directed	107,614	13,673,453	Social circles from Google+
ego-Twitter	Directed	81,306	1,768,149	Social circles from Twitter

Figure 4.2: Stanford Large Network Dataset Collection - SNAP

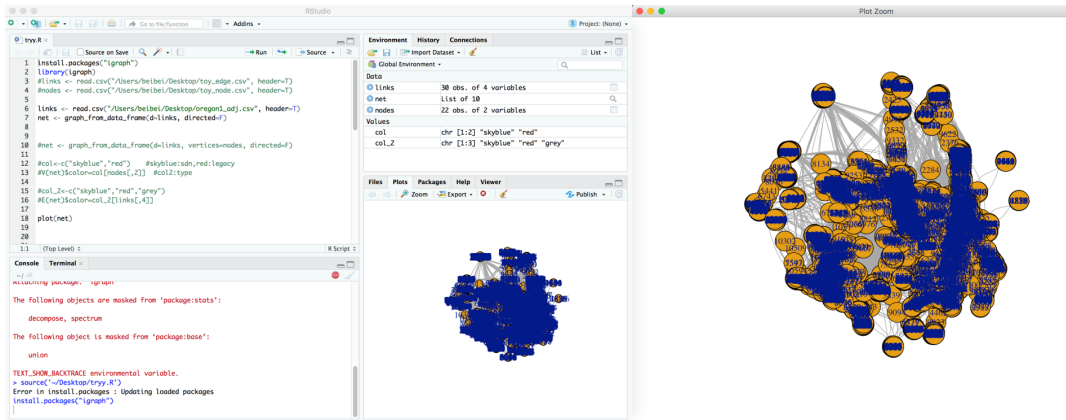


Figure 4.3: Network Topology Data Imported From Dataset

SDN switch are in blue. Links between legacy router and legacy routers are in red. Links with different types of ends are in gray.

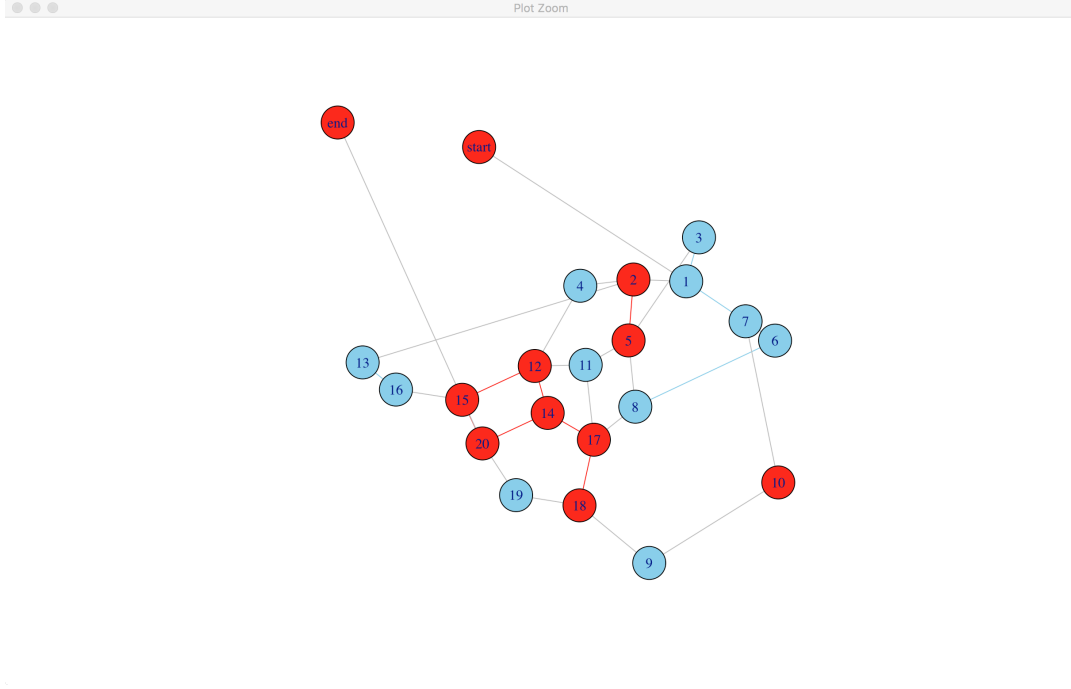


Figure 4.4: Network Topology with Hybrid Deployment

4.3 System Setting

We implement the design with network emulator Mininet [13] and SDN controller ONOS [2]. Mininet sets up the data-plane topology and is responsible for carrying out the forwarding rules, while ONOS is the control plane determining the forwarding logic and behaviors. The Mininet in virtualBox and the ONOS are depicted in the following figure 4.5 and figure 4.6.

Figure 4.5 shows the network emulator Mininet embedded in the virtualBox. The Ethernet interface eth0 is configured as a bridged adapter, which is allocated an IP address. The interface eth1 is configured as host-only adapter, which is to communicate with the host machine only, as we need to use shell (ssh command) to log into Mininet from the host machine for graphics (e.g., xterm, miniedit applications).

The left part of figure 4.6 is the result after compiling the ONOS source code and

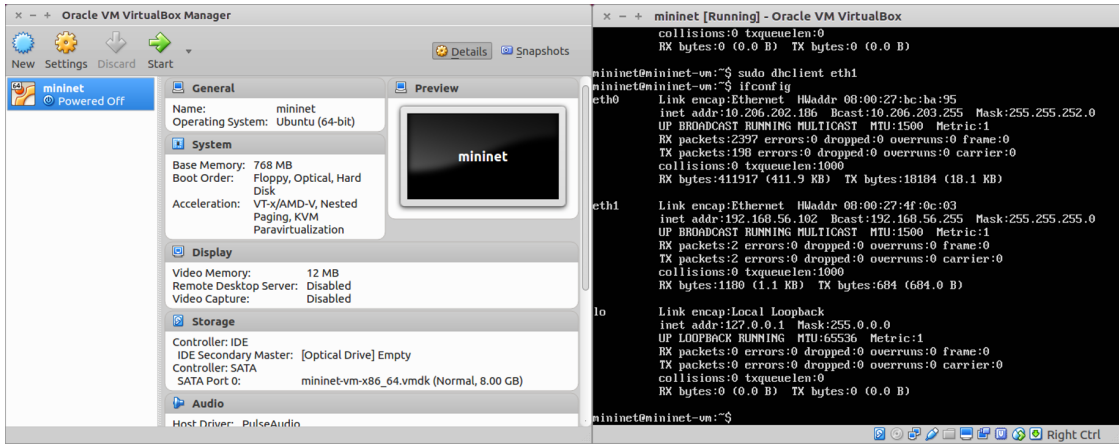


Figure 4.5: Mininet in VirtualBox

installing together with Karaf (ONOSKaraf clean install). The right part of figure 4.6 is the interactive terminal of ONOS.

Figure 4.7 is the initial topology. There is one controller, 20 SDN switches running

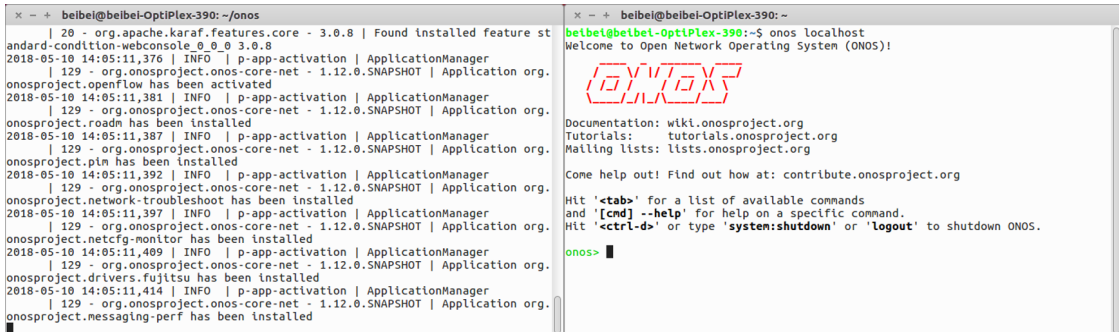


Figure 4.6: Open Networking Operating System (ONOS)

OpenFlow protocol, and two hosts, representing the power SCADA control center and water SCADA control center.

The legacy routers run the BGP part of Quagga. There is a converting module SDN-IP, within which there is a BGP speaker, who can talk using BGP with these routers to exchange traffic between the different SDN islands. Controller ONOS cannot see either these hosts or legacy switches directly. Transit traffic (traffic from one BGP

peer outside the local SDN network that traverses a local SDN network and goes to another BGP peer) are proactively installed by the SDN-IP application.

For a given topology, the SDN deployment rate is defined as the ratio of number of

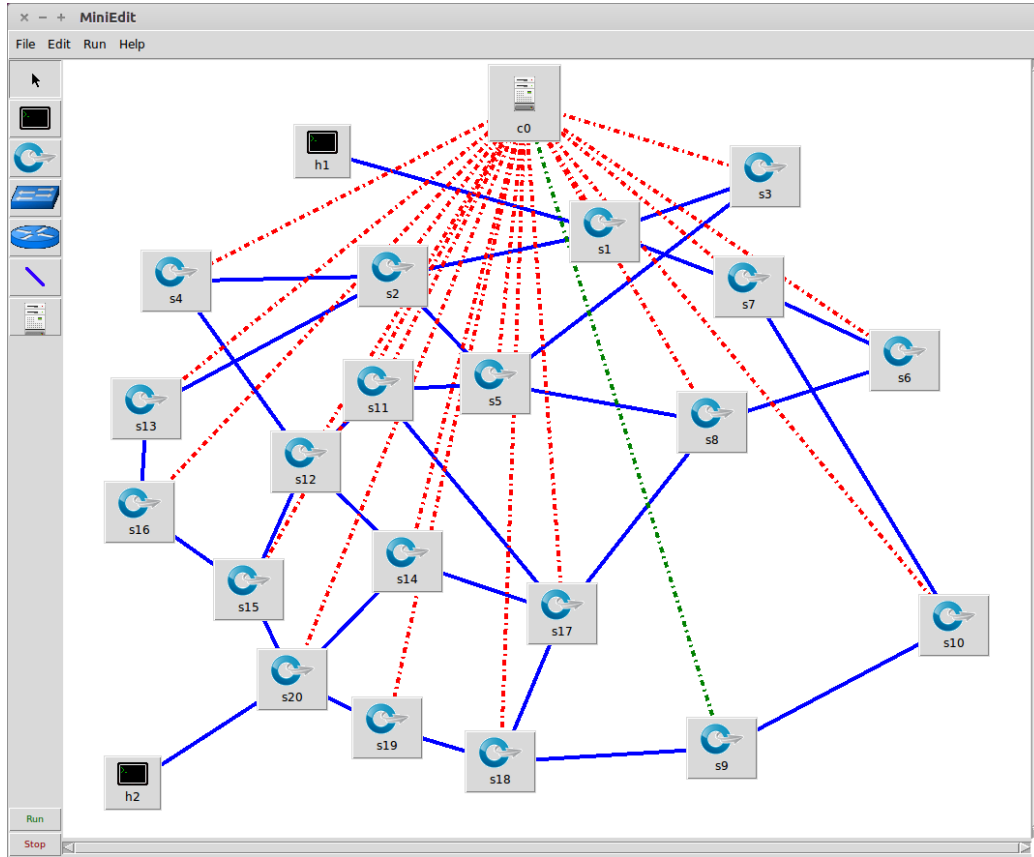


Figure 4.7: Emulator Topology

SDN switches to the number of total switches (routers). Figure reffig:rate shows the SDN deployment rates varying from 95% to 70%.

There are 3 types of links between switches:

1. SDN switch - SDN switch (homogeneous)
2. legacy router - legacy router (homogeneous)
3. SDN switch - legacy router (heterogenous)

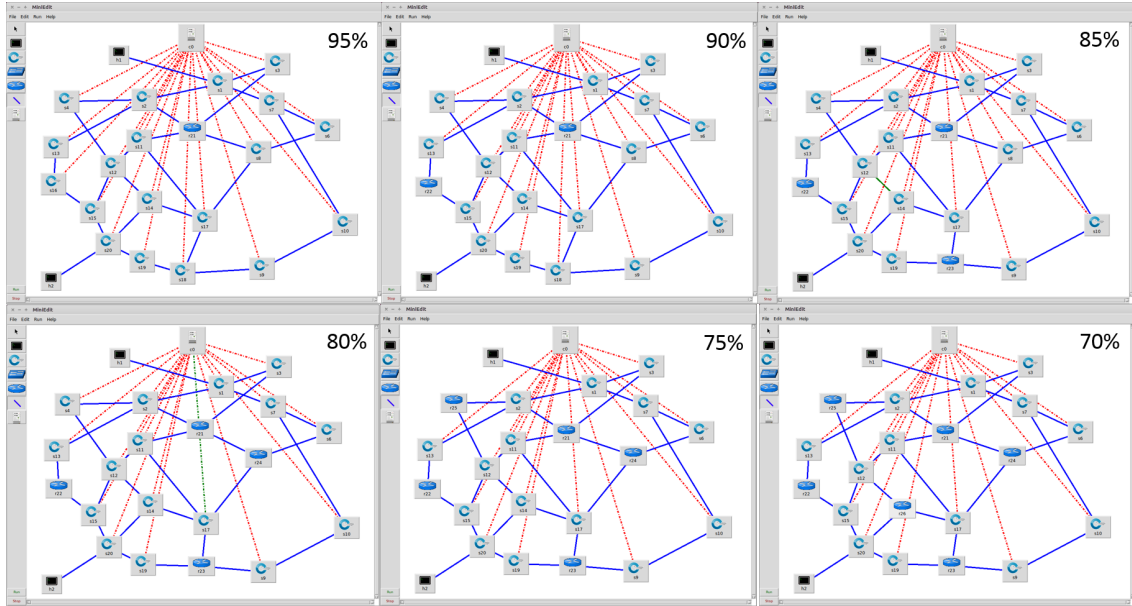


Figure 4.8: Different SDN Deployment Rate

If we are sending an Internet Control Messages Protocol (ICMP) message from host 1 to host 2, there are several possible paths:

1. h1-s1-s2-s4-s12-s15-s20-h2
2. h1-s1-s7-s10-s9-s18-s19-s20-h2
3.

Different paths have different costs and performances. The metrics we are evaluating are:

1. End-to-end delay: a time stamp is recorded at the beginning; another time stamp is recorded at the end. Then end-to-end delay is calculated.
2. Throughput: Each link has a real-time bandwidth. For a path (a few links concatenated), throughput is defined as the smallest bandwidth along the path.

The topology discovered by the LLDP probes can be directed to a Graphical User Interface (GUI). Figure 4.9 is the GUI of the ONOS controller. All SDN switches can be detected by LLDP probes and reported to the central controller.

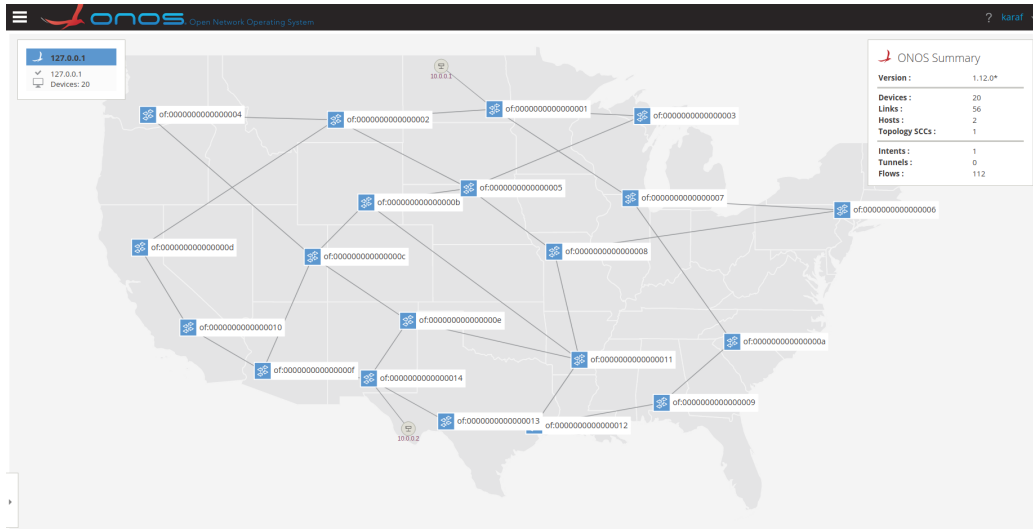


Figure 4.9: IP Addresses of Hosts

4.4 Program Architecture

The program consists of a few blocks:

1. packet generation block

- (a) ICMP packets

The ping command generates ICMP packets. It is a protocol used by network devices to send error messages and operational information indicating that a host or router could be reached or not. Testing the connection with ICMP packets is easy. However, the amount of data is less than Scapy (elaborated later) or Iperf.

The details of captured packets are shown in figure 4.10. The highlighted

lines are the type of packets we are interested in.

No.	Time	Source	Destination	Protocol	Length	Info
17	6.299509000	02:eb:9f:67:c9:42	OpenNetw_00:00:01	LLDP	81	Chassis Id = 00:00:00:00:00:05 Port Id = TTL = 120
18	6.299640000	02:eb:9f:67:c9:42	Broadcast	0x8942	81	Ethernet II
19	9.300574000	02:eb:9f:67:c9:42	OpenNetw_00:00:01	LLDP	81	Chassis Id = 00:00:00:00:00:01 Port Id = TTL = 120
20	9.300671000	02:eb:9f:67:c9:42	Broadcast	0x8942	81	Ethernet II
21	9.302808000	02:eb:9f:67:c9:42	OpenNetw_00:00:01	LLDP	81	Chassis Id = 00:00:00:00:00:03 Port Id = TTL = 120
22	9.302887000	02:eb:9f:67:c9:42	Broadcast	0x8942	81	Ethernet II
23	9.398956000	02:eb:9f:67:c9:42	OpenNetw_00:00:01	LLDP	81	Chassis Id = 00:00:00:00:00:05 Port Id = TTL = 120
24	9.399102000	02:eb:9f:67:c9:42	Broadcast	0x8942	81	Ethernet II
25	11.740676000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
26	11.739583000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
27	11.738924000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
28	11.739253000	10.0.0.5	10.0.0.6	ICMP	98	Echo (ping) request id=0x0e8b, seq=1/256, ttl=64
29	11.741367000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
30	11.737488000	10.0.0.5	10.0.0.6	ICMP	98	Echo (ping) request id=0x0e8b, seq=1/256, ttl=64
31	11.740235000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
32	11.737670000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
33	12.400531000	02:eb:9f:67:c9:42	OpenNetw_00:00:01	LLDP	81	Chassis Id = 00:00:00:00:00:01 Port Id = TTL = 120
34	12.400639000	02:eb:9f:67:c9:42	Broadcast	0x8942	81	Ethernet II
35	12.402677000	02:eb:9f:67:c9:42	OpenNetw_00:00:01	LLDP	81	Chassis Id = 00:00:00:00:00:03 Port Id = TTL = 120
36	12.402767000	02:eb:9f:67:c9:42	Broadcast	0x8942	81	Ethernet II
37	12.498621000	02:eb:9f:67:c9:42	OpenNetw_00:00:01	LLDP	81	Chassis Id = 00:00:00:00:00:05 Port Id = TTL = 120
38	12.498750000	02:eb:9f:67:c9:42	Broadcast	0x8942	81	Ethernet II
39	12.738985000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
40	12.738026000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
41	12.738590000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
42	12.739736000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6
43	12.737463000	06:a7:07:77:30:e9	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.6

Filter: icmp Expression... Clear Apply Save						
No.	Time	Source	Destination	Protocol	Length	Info
28	11.739253000	10.0.0.5	10.0.0.6	ICMP	98	Echo (ping) request id=0x0e8b, seq=1/256, ttl=64
30	11.737488000	10.0.0.5	10.0.0.6	ICMP	98	Echo (ping) request id=0x0e8b, seq=1/256, ttl=64
183	36.808954000	10.0.0.1	10.0.0.6	ICMP	98	Echo (ping) request id=0x0e9c, seq=1/256, ttl=64
186	36.806819000	10.0.0.1	10.0.0.6	ICMP	98	Echo (ping) request id=0x0e9c, seq=1/256, ttl=64
227	46.860200000	10.0.0.3	10.0.0.6	ICMP	98	Echo (ping) request id=0x0ea3, seq=1/256, ttl=64
265	56.896624000	10.0.0.2	10.0.0.6	ICMP	98	Echo (ping) request id=0x0eab, seq=1/256, ttl=64
269	56.897933000	10.0.0.2	10.0.0.6	ICMP	98	Echo (ping) request id=0x0eab, seq=1/256, ttl=64
302	66.937156000	10.0.0.4	10.0.0.6	ICMP	98	Echo (ping) request id=0x0eb2, seq=1/256, ttl=64
305	66.937795000	10.0.0.4	10.0.0.6	ICMP	98	Echo (ping) request id=0x0eb2, seq=1/256, ttl=64

Figure 4.10: Packet Flows in the Networks

(b) Scapy

Scapy generates any type of packet, IP, TCP, ICMP, etc. They can come from different layers. The forwarding rules running on the network layer will check for the IP packets, making decisions based on IP header; while forwarding rules on transport layer will check for TCP/UDP packets, looking at TCP/UDP headers; forwarding rules on the application layer will check for ICMP packets, interested in application layer header or content of the messages.

The bandwidth of each link are measured. Result of a part of the network is shown in figure 4.11.

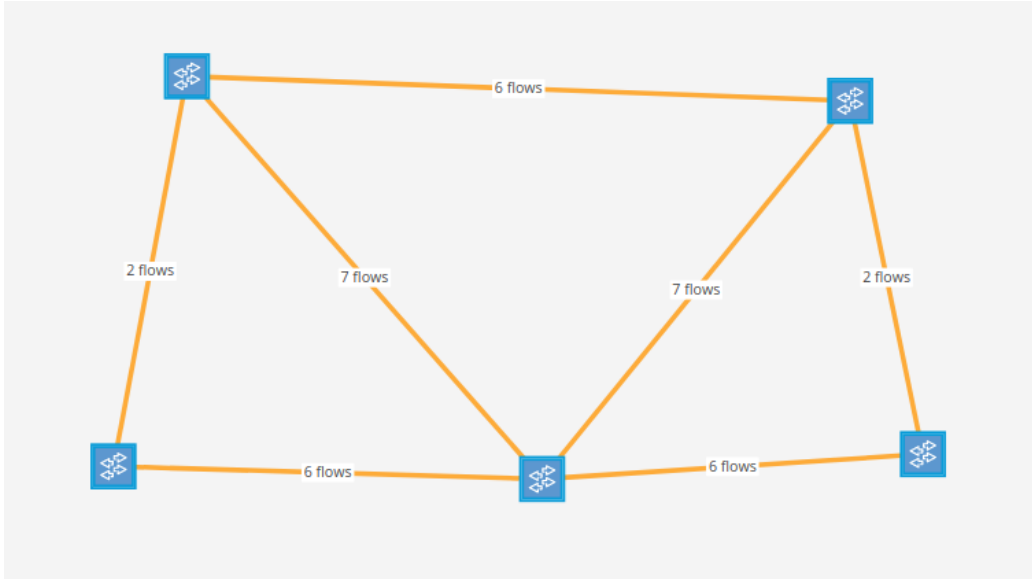


Figure 4.11: All Flows Going through a Given SDN Switch

2. network programming block

(a) general forwarding sub-block

There are two general ways of programming the network: flow entry table and intent.

Flow entries are forwarding rules programmed manually into switches or injected by controllers.

The Intent Framework depicted in figure 4.12 is a subsystem that allows applications to specify their network control desires in the form of a policy rather than mechanism.

Intents are more flexible than flow entries. Intents track the status of the network, ready to reconfigure the network whenever there is a link break-

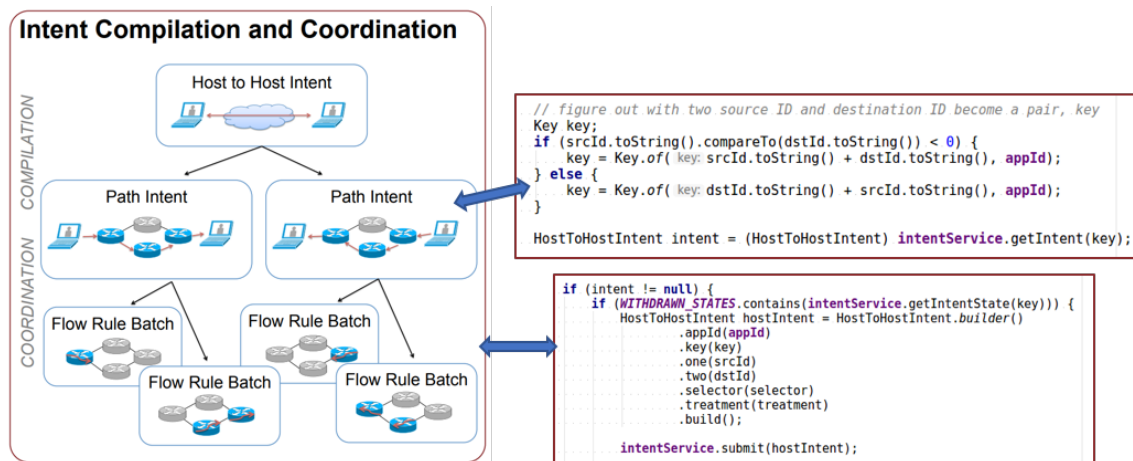


Figure 4.12: Intent Conveys Policies to Data Plane

down. Intent can reroute to an alternative path if needed.

(b) SDN-IP sub-block

The components for the SDN-IP block are shown in figure 4.13.

3. performance evaluation block

(a) end-to-end delay:

It is a metric measuring the real-time capability of the middleware, as the critical infrastructure needs immediate notice and response.

(b) throughput:

It is a bandwidth-related metric to measure the amount of traffic that the middleware can handle within a period of time.

4. SDN-IP Module block

An SDN-IP module acts as a transit interconnecting SDN-controlled networks and IP networks. Each external legacy network (IP network) is a different domain, which interfaces with our SDN network through its BGP-speaking border

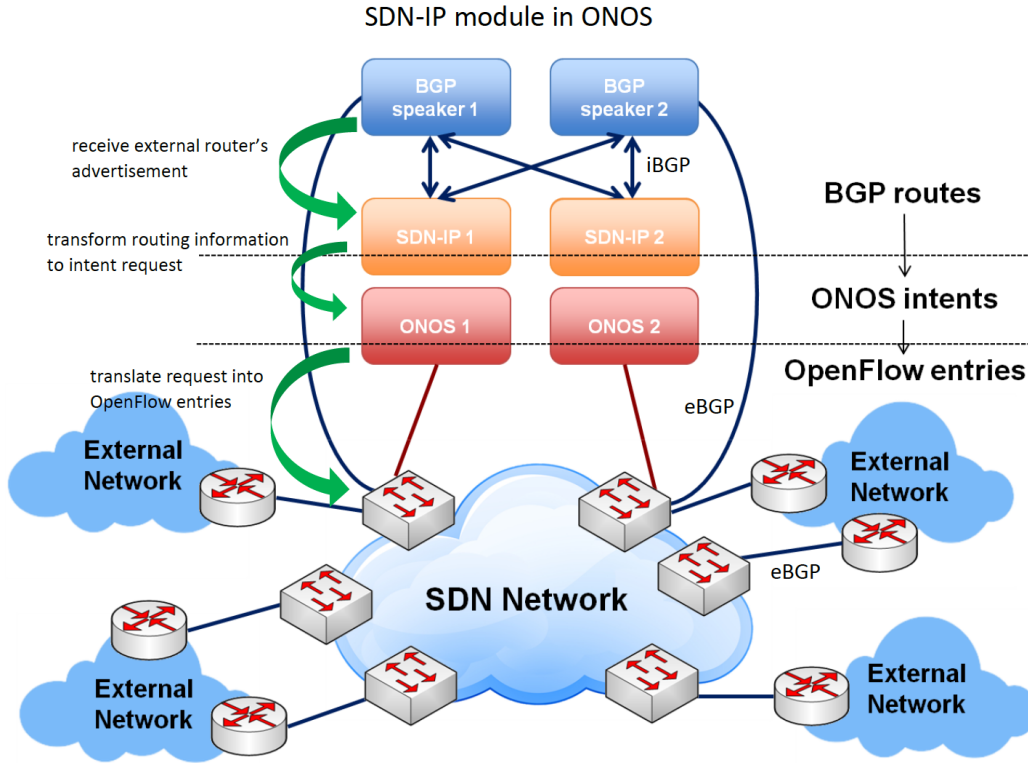


Figure 4.13: Components for SDN-IP

routers [22].

Within the SDN-IP network, there are one or more internal BGP speakers. The BGP speakers can be existing BGP routers or any software that implements BGP. The speakers use eBGP to exchange BGP routing information with the border routers of the adjacent external networks, and iBGP to propagate that information among themselves and to the SDN-IP application instances [22].

The routes advertised by the external border routers belonging to external networks are received by the BGP speakers within the SDN-IP network, processed, and eventually re-advertised to the other external networks. The routes are processed according to the normal BGP processing and routing policies. Similarly, those routes are also advertised to the SDN-IP application instances, which act

as iBGP peers. The best route for each destination is selected by the SDN-IP application according to the iBGP rules, and translated into an ONOS Application Intent Request. ONOS translates the Application Intent Request into forwarding rules in the data plane. Those rules are used to forward the transit traffic between the interconnected IP networks [22].

The major function blocks are shown in figure 4.14.

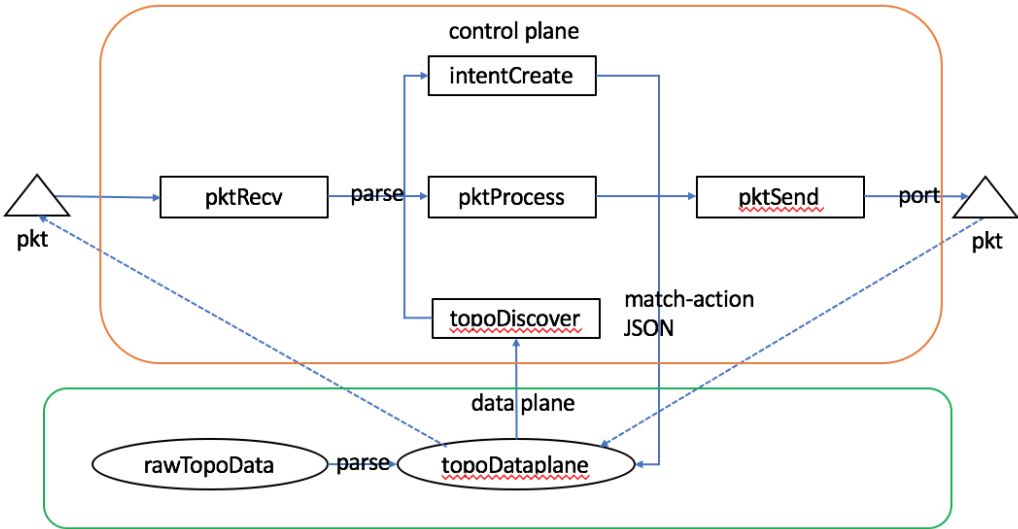


Figure 4.14: Diagrams for the Program

4.5 Communication between Emulators and Middleware

4.5.1 Multi-threaded Communication

To guarantee the real-time data transmission, a multithreaded program is developed and encapsulated as a Communication module (a Python class). One SCADA system may need to communicate to one or more other SCADA systems. Once the

emulation program instantiates the class, simulation results can be transmitted to the target receivers reliably in real time. At the same time, each receiver listens to any incoming connection. Operations of the multithreaded program are shown in figure 4.15.

In Figure 4.15, there are four threads in the communication module of utility 1:

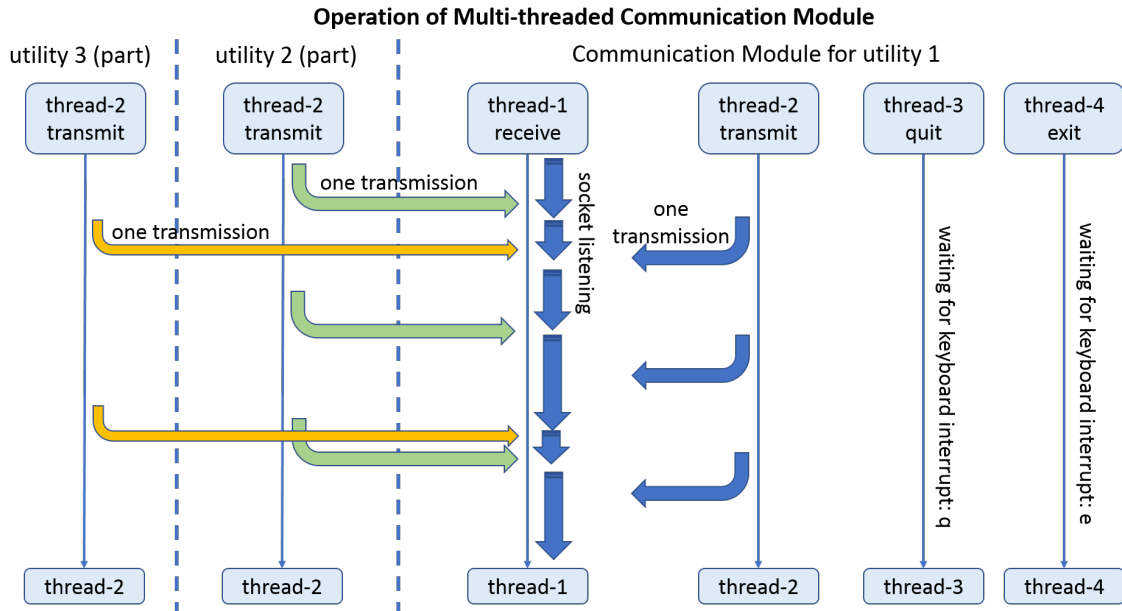


Figure 4.15: Multithreaded Communication Module

1. Thread-1 is a receiving thread, listening constantly to a designated port; once a connection comes, it forks another new thread to take care of the data reception task, the original thread keeps listening.
2. Thread-2 is a transmit thread, initiating a data transmission to a specific target, by specifying the IP address and port of the target. If the file to transmit is not ready, thread-2 will keep trying fetching. If the target is not responding,

thread-2 will try to connect repeatedly after a given interval, until connection is established finally.

3. Thread-3 is a quit thread. It is for human intervention in case one wants to stop transmitting files to the target utility. Thread-3 is routinely querying from the console/keyboard input, generating interrupts whenever there is any input. If the captured input is not the desired one for quit, it will prompt an error and keep querying.
4. Thread-4 is an exit thread, waiting for console/keyboard interrupt, similar to thread-3. The major difference from thread-3 is that thread-4 is to stop listening on a port and close the server, thus it stops the reception.

4.5.2 *Relation to Emulator*

The block diagram of the communication module is shown in figure 4.16. Each block in the diagram represents a Python program. It comes in such a complicated manner because the power and water emulators run on the Windows operating system while the middleware runs on the Linux operating system. Some control and convert processes need to be carried out. It is almost symmetric so let us take the power side for example.

The result generated from power simulations and optimizations are named `powerOutFile_0.txt`, `powerOutFile_1.txt`, etc. Then the Python script `comm_win_power.py` handles the file transmission, forwarding the files one after another to module `mininet.py` in middleware. Middleware is a container divided into sub-containers by different namespaces. `mininet1.py` runs on the hosting container, while `host1.py` and `host2.py` are two emulated nodes running on sub-containers. Data are transmitted from the source host, through the emulated networks, reaching the destination host.

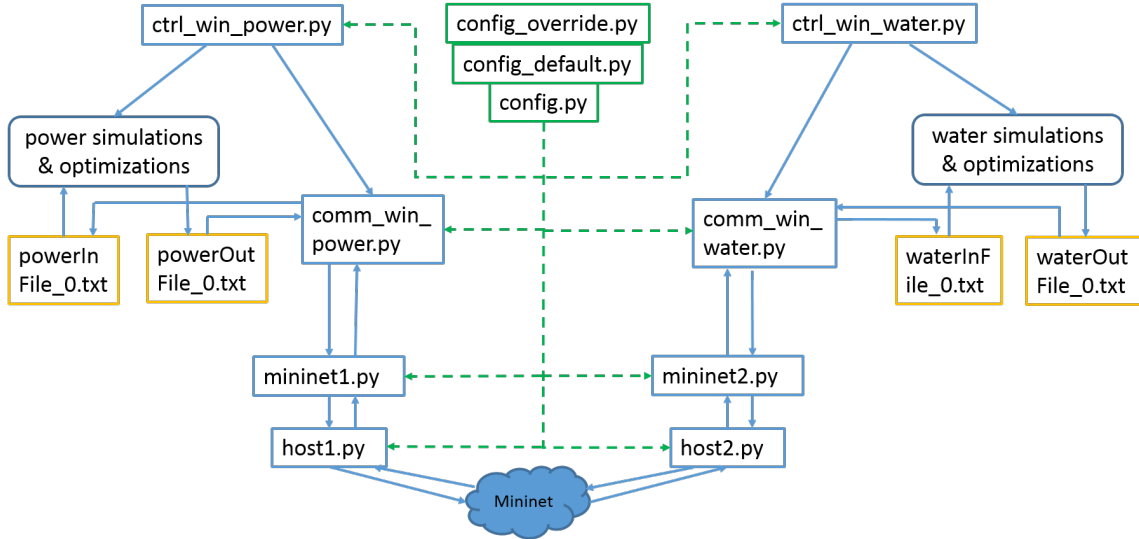


Figure 4.16: Block Diagram of Communication Module

All the block programs are controlled by `ctrl_win_power.py`, which governs the time sequence of operations. All programs are configured by `config.py` files (`config_default.py` is the default setting, `config_override.py` is the changes on a particular run).

4.5.3 Cross Platform Features

The result data are generated at the emulators, passed through SCADA gateways to the backbone network. Then, after traversing the backbone, it ends up with another gateway and reaches the destination emulator.

The emulators run on the Windows operating system, while the middleware runs on the Linux operating system. Therefore, to cater for cross-platform needs, the communication module adopts some features so that it can be applied to Windows/Linux/Mac systems.

Features include but not limited to: Each system has its own buffer. To avoid buffer overflow, data are read and written line-by-line. Every time the communication mod-

ule runs, it uses the socket API to detect the host IP, for better portability. In this way, the communication module can be reused easily by embedding into emulation code. Another programmer can instantiate the class and use it for efficient data transmissions.

4.6 Apply the Module to the Platform

ONOS application is built using the standard OSGi bundle management services. After the function module is developed, it should be installed and activated on the ONOS platform.

The activation of the module is shown in the figure 4.17.

4.7 Varied Ratio of SDN Deployment

Seen from the SDN controller, the topology of SDN deployment with varied ratio are illustrated in figure 4.18. It shows that the higher SDN penetration, the less conversion from SDN switches to legacy routers are needed, which makes the data transmission more efficient.

4.8 Performance Evaluations

For the performance evaluation:

The end-to-end delay of hosts is one of the most important factors for a SCADA communication system. A smaller delay will not only benefit the function of critical infrastructure but also provide a better user experience.


```

x - + beibei@beibei-OptiPlex-390: ~/onos
beibei@beibei-OptiPlex-390:~$ cd onos
beibei@beibei-OptiPlex-390:~/onos$ onos localhost
Welcome to Open Network Operating System (ONOS)!

  ONOS

Documentation: wiki.onosproject.org
Tutorials:    tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos> apps -s -a
* 17 org.onosproject.drivers          1.12.0.SNAPSHOT Default Drivers
* 31 org.onosproject.optical-model    1.12.0.SNAPSHOT Optical Network Model
* 40 org.onosproject.hostprovider     1.12.0.SNAPSHOT Host Location Provide
r
* 51 org.onosproject.mobility         1.12.0.SNAPSHOT Host Mobility
* 52 org.onosproject.fwd              1.12.0.SNAPSHOT Reactive Forwarding
* 55 org.onosproject.lldpprovider     1.12.0.SNAPSHOT LLDP Link Provider
* 95 org.onosproject.openflow-base   1.12.0.SNAPSHOT OpenFlow Base Provide
r
* 96 org.onosproject.openflow        1.12.0.SNAPSHOT OpenFlow Provider Sui
te
* 136 org.onosproject.proxyarp       1.12.0.SNAPSHOT Proxy ARP/NDP
onos>

```

Figure 4.17: Install and Activate the Applications in ONOS

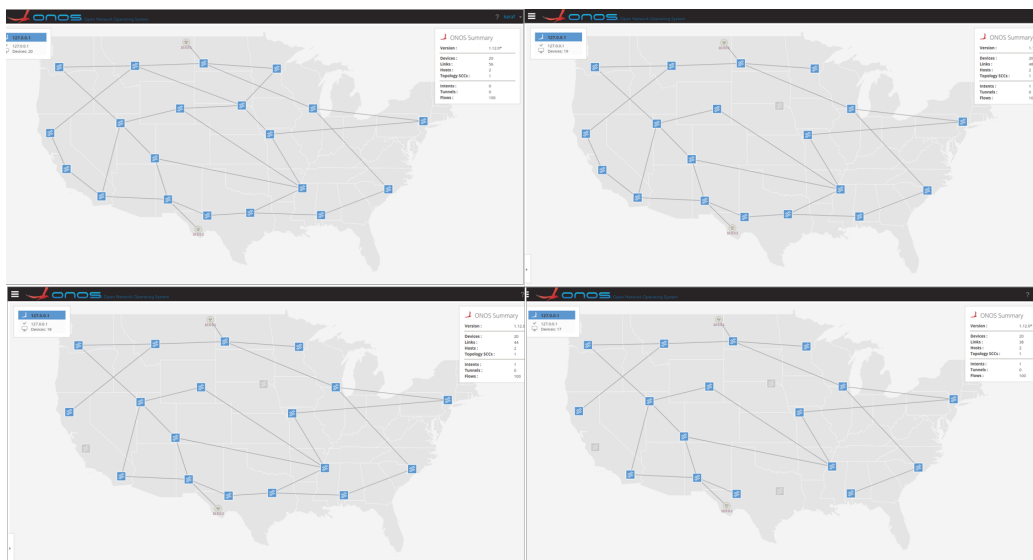


Figure 4.18: Varied Ratio of SDN Deployment

The results can be seen in figure 4.19. It is easily seen that with the decrement of SDN deployment, the delays are increasing gradually.

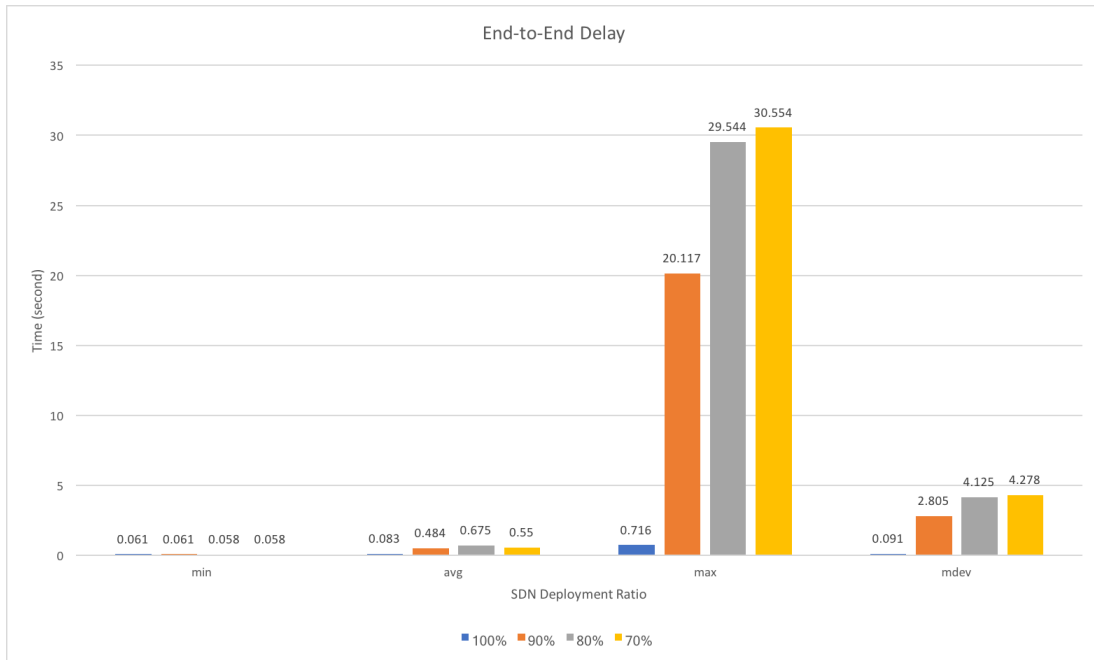


Figure 4.19: Experiment Results:End-to-End Delay

The throughput of both the OpenFlow Controller and OpenFlow switches is a key performance indicator for the Control Plane operation [7]; the number of flow entries in the table of the OpenFlow switch can impact the forwarding efficiency of the switch, as every incoming packet has to be matched against entries in the flow table [7].

The throughput against SDN deployment ratio is illustrated in Figure 4.20. As we can see from the figure, the reduced deployment of SDN will negatively affect the throughput of the network. In conclusion, more SDN should be adopted if conditions permit.

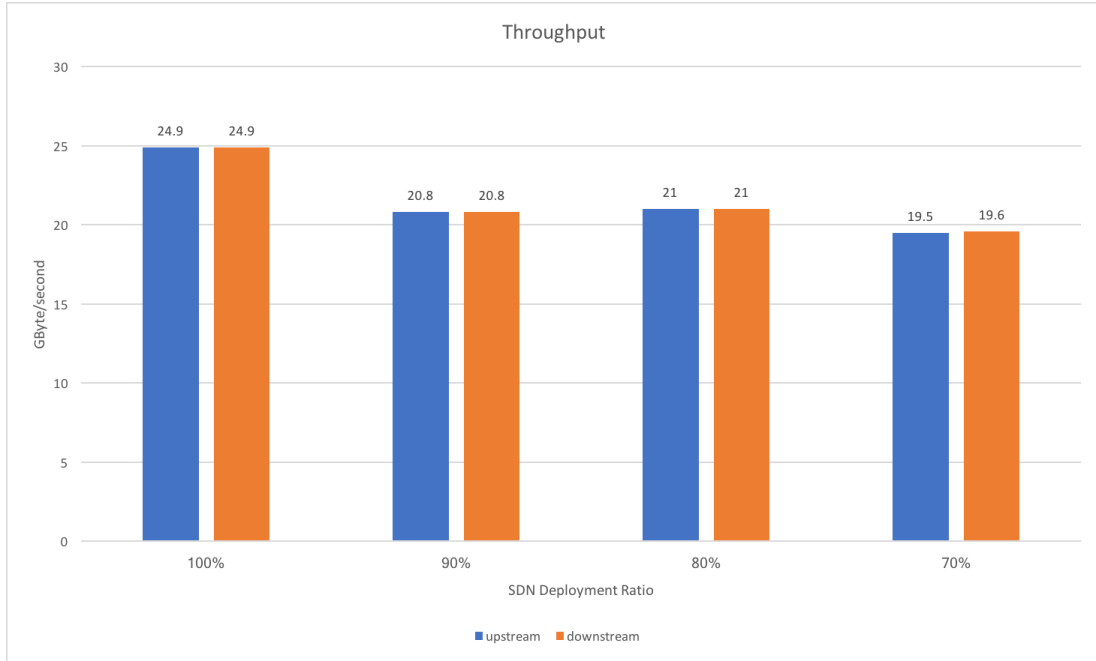


Figure 4.20: Experiment Results:Throughput

4.9 Summary

In this chapter, the implementation details of the proposed middleware were presented. Firstly, we imported network data from Stanford Large Network Dataset Collection, which is a real-world data set. However, for the ease of clearer presentation, we used a simpler topology in the experiment. Secondly, topology information was put into network emulator Mininet. Thirdly, we programmed the forwarding behaviors as intents and injected into controller and activated it on the ONOS platform. Lastly, network functions were tested and performances are evaluated.

CONCLUSION AND FUTURE WORK

5.1 Conclusion

For the critical infrastructure communications, it is a trend to gradually move from dedicated networks to public networks (e.g., Internet or WANs). Instead of using proprietary devices and wires, critical infrastructures will choose to embrace the new method of leasing them from the service providers. The trend of going beyond the local networks facilitates the collaboration of two or more SCADA systems. In this work, we provide an insight on this trend.

In this work, an SDN-assisted middleware is designed and built to facilitate the cyberphysical (power SCADA systems and water distribution SCADA systems) co-simulations. As SDN is a software-centric approach for communication networks, the packet forwarding mechanism is determined by the software in the centralized controller. Programmability of networks simplifies the configuration process and provides a more reliable and flexible communication.

To improve the network reliability and efficiency of SDN-based forwarding, we utilized multipath transmission. The result shows that the alternative path guarantees a better connectivity and thus a lower packet loss rate.

A full upgrade of network devices may result in huge capital expenditures for an operator. A gradual deployment approach is proposed. Hybrid deployment of SDN switches and legacy routers is adopted and discussed in this work. The results show that a higher SDN deployment rate results in a better performance of data transmissions.

The proposed middleware is implemented with the prevailing open source controller platform ONOS. A multithreaded real-time communication module to connect the SCADA system gateway is developed. Relevant codes are tested, commented, and documented before uploading to GitHub for reproduction purpose or potential future usage.

5.2 Future Work

In this work, we built a middleware to interconnect two or more SCADA systems. There are some other opportunities or open problems to discuss in future.

As a large number of requests will be handled by a single controller, for better network scalability, more controllers can be utilized. The problem of the coordination among controllers is a direction of further research.

In this work, only the node or link breakdown in physical networks was considered. There are other categories of accidents or attacks worth considering in future. For example, compromised network switches, broken grid devices, or missing SDN controllers [5]. New cyber-security solutions are needed immediately to protect the SCADA communication systems.

Many aspects of the network operation besides resilience deserve our attentions. More QoS metric options should be provided in the middleware. For example, efficient traffic engineering, load balancing, algorithm convergence speed, etc. Those problems can be addressed by utilizing SDN, because SDN provides the central controller with a global view of the topology and traffic.

Apart from communications between SCADA systems, SDN-assisted middleware can also be used between enterprise networks, data centers, or service provider networks. It virtualizes the resource allocation process, making it reliable and efficient. Solu-

tions adapted to those specific scenarios should be developed.

The control plane provides APIs to various applications. Therefore, it is convenient to develop third party software to employ some sophisticated functions. The applications can contribute to open source communities.

5.3 Summary

In this chapter, we conclude our work and identify some possibilities for future research. Various interesting problems or topics can be extended in the future. We believe the concept of software-defined will enable innovations in many areas beyond the problem discussed in this thesis.

BIBLIOGRAPHY

- [1] Akyildiz, I. F., A. Lee, P. Wang, M. Luo and W. Chou, “Research challenges for traffic engineering in software defined networks”, *IEEE Network* **30**, 3, 52–58 (2016).
- [2] Berde, P., M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O’Connor, P. Radoslavov, W. Snow *et al.*, “Onos: towards an open, distributed sdn os”, in “Proceedings of the third workshop on Hot topics in software defined networking”, pp. 1–6 (ACM, 2014).
- [3] Caesar, M. and J. Rexford, “Bgp routing policies in isp networks”, *IEEE network* **19**, 6, 5–11 (2005).
- [4] Csanyi, E., “3 generations of scada system architectures you should know about”, <http://electrical-engineering-portal.com/three-generations-of-scada-system-architectures>, april 2, 2013 (2013).
- [5] Dong, X., H. Lin, R. Tan, R. K. Iyer and Z. Kalbarczyk, “Software-defined networking for smart grid resilience: Opportunities and challenges”, in “Proceedings of the 1st ACM Workshop on Cyber-Physical System Security”, pp. 61–68 (ACM, 2015).
- [6] Feamster, N., J. Rexford and E. Zegura, “The road to sdn: an intellectual history of programmable networks”, *ACM SIGCOMM Computer Communication Review* **44**, 2, 87–98 (2014).
- [7] Giotis, K., C. Argyropoulos, G. Androulidakis, D. Kalogeras and V. Maglaris, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments”, *Computer Networks* **62**, 122–136 (2014).
- [8] Jain, S., A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, “B4: Experience with a globally-deployed software defined wan”, in “ACM SIGCOMM Computer Communication Review”, vol. 43, pp. 3–14 (ACM, 2013).
- [9] Jararweh, Y., M. Al-Ayyoub, E. Benkhelifa, M. Vouk, A. Rindos *et al.*, “Sdiot: a software defined based internet of things framework”, *Journal of Ambient Intelligence and Humanized Computing* **6**, 4, 453–461 (2015).
- [10] Katiyar, R., P. Pawar, A. Gupta and K. Kataoka, “Auto-configuration of sdn switches in sdn/non-sdn hybrid network”, in “Proceedings of the Asian Internet Engineering Conference”, pp. 48–53 (ACM, 2015).
- [11] Kotronis, V., A. Gämperli and X. Dimitropoulos, “Routing centralization across domains via sdn: A model and emulation framework for bgp evolution”, *Computer Networks* **92**, 227–239 (2015).

- [12] Lakshminarayanan, K., I. Stoica, S. Shenker and J. Rexford, *Routing as a Service* (Computer Science Division, University of California Berkeley, 2004).
- [13] Lantz, B., B. Heller and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks”, in “Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks”, p. 19 (ACM, 2010).
- [14] Lin, D., D. Hui, W. Wu, T. Liu, Y. Yang, Y. Wang, J. C. Lui, G. Zhang and Y. Li, “On the feasibility of inter-domain routing via a small broker set”, in “Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on”, pp. 2031–2038 (IEEE, 2017).
- [15] Schaller, S. and D. Hood, “Software defined networking architecture standardization”, *Computer Standards & Interfaces* **54**, 197–202 (2017).
- [16] Stouffer, K., J. Falco and K. Scarfone, “Guide to industrial control systems (ics) security”, NIST special publication **800**, 82, 16–16 (2011).
- [17] Wang, F., Z. M. Mao, J. Wang, L. Gao and R. Bush, “A measurement study on the impact of routing events on end-to-end internet path performance”, in “ACM SIGCOMM Computer Communication Review”, vol. 36, pp. 375–386 (ACM, 2006).
- [18] Wang, W., W. He and J. Su, “Boosting the benefits of hybrid sdn”, in “Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on”, pp. 2165–2170 (IEEE, 2017).
- [19] Wikipedia, “Autonomous system (internet)”, [https://en.wikipedia.org/wiki/Autonomous_system_\(Internet\)](https://en.wikipedia.org/wiki/Autonomous_system_(Internet)) (2018).
- [20] Wikipedia, “Network discovery”, <https://wiki.onosproject.org/display/ONOS/Network+Discovery#NetworkDiscovery-SDNislandsandlegacynetworks> (2018).
- [21] Wikipedia, “Scada”, <https://en.wikipedia.org/wiki/SCADA> (2018).
- [22] Wikipedia, “Sdn-ip architecture”, <https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture> (2018).
- [23] Zhang, S., J. Lee, C. Tepedelenioglu and A. Spanias, “Distributed estimation of the degree distribution in wireless sensor networks”, in “2016 IEEE Global Communications Conference (GLOBECOM)”, pp. 1–6 (2016).
- [24] Zhang, S., C. Tepedelenioglu, J. Lee, H. Braun and A. Spanias, “Cramer-rao bounds for distributed system size estimation using consensus algorithms”, in “2016 Sensor Signal Processing for Defence (SSPD)”, pp. 1–5 (2016).
- [25] Zhang, S., C. Tepedelenioglu and A. Spanias, “Distributed center and coverage region estimation in wireless sensor networks using diffusion adaptation”, in “2017 51st Asilomar Conference on Signals, Systems, and Computers”, pp. 1353–1357 (2017).

- [26] Zhang, S., C. Tepedelenlioglu, A. Spanias, M. Banavar and W. Tranter, *Distributed Network Structure Estimation Using Consensus Methods* (Morgan and Claypool, 2018), URL <https://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=8307808>.
- [27] Zhang, S., C. Tepedelenliolu, M. K. Banavar and A. Spanias, “Distributed node counting in wireless sensor networks in the presence of communication noise”, *IEEE Sensors Journal* **17**, 4, 1175–1186 (2017).
- [28] Zhang, S., C. Tepedelenliolu and A. Spanias, “Distributed network center and size estimation”, *IEEE Sensors Journal* (2018).
- [29] Zhang, S., C. Tepedelenliolu, M. K. Banavar and A. Spanias, “Max consensus in sensor networks: Non-linear bounded transmission and additive noise”, *IEEE Sensors Journal* **16**, 24, 9089–9098 (2016).