Using Route Forwarding State to Identify Routing Errors in Mission-Critical, Military Data Networks

by

Philip Cartier Drew

B.S., University of Colorado at Boulder, 2004

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Telecommunications

2017

This thesis entitled:
Using Route Forwarding State to Identify Routing Errors in Mission-Critical, Military Data Networks
written by Philip Cartier Drew
has been approved for the Department of Telecommunications

Dr. Levi Perigo

Prof. Joe McManus

Prof. Jose Santos

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Drew, Philip Cartier (M.S., Telecommunications)

Using Route Forwarding State to Identify Routing Errors in Mission-Critical, Military Data Networks

Thesis directed by Dr. Levi Perigo

This research verified route forwarding state information from an example, mission-critical, military data network against a network architecture definition in order to aid inexperienced network administrators in identifying routing errors. Aiding inexperienced administrators is important because network administrators in the military vary widely in experience and it is more likely that mission-critical, military data networks are administered by people who lack experience.

As part of this research, an Architecture Definition Language (ADL) was developed to specify physical topology and data-link attributes of a network. That architecture definition was then used to define a model of the network using graph theory tools. Next, a web-based visualization of the architecture model was developed that included an interactive form to evaluate the shortest path between subnetworks. The shortest path between subnetworks was calculated on the architecture model, inspected on the live network, and overlaid on the web-based visualization to depict the inconsistencies for an inexperienced network administrator.

With the tool developed in this research, an experienced network administrator can use the developed ADL to describe how the network should work to network administrators at any experience level. Using the architecture definition, inexperienced network administrators can identify routing errors by visualizing how a live network compares to the architected network. Quickly identifying and resolving routing errors in mission-critical, military data networks enhances the operational effectiveness of the military.

Dedication

To Lisa, Nathan, and Mackenzie.

Acknowledgements

Above all, I am grateful to Lisa for supporting and encouraging me through six long years in this program. Thank you for helping me become the best version of myself.

Thank you to Nathan and Mackenzie for understanding when I needed to work on homework or research and couldn't give you my undivided attention.

I would also like to thank Dr. Levi Perigo for his advice and encouragement during my thesis research. I finally reached the finish line.

Contents

Figures

Chapter 1

Introduction

Although organizations representing the United States military are credited with developing networks and protocols that led to the modern-day Internet, those same organizations and experienced professionals do not plan, install, operate, maintain, and defend mission-critical, military data networks in today's environment [4]. Network administrators in the military vary widely in training and experience [33]; ranging from less than a year of experience with any form of data network to decades of experience architecting and managing complex networks in hostile environments, supporting mission-critical requirements [16]. Because of this disparity in experience levels, and the fact that nearly half (43.6%) of the active duty military force is within the first four pay grades [3, p. 15], it is more likely that mission-critical, military data networks are administered by people who lack experience. The combination of increasing complexity of data networks [37, Ch. 5] and increasing reliance on data services within the military [37, Ch. 5] makes having experienced network administrators critical [16].

Network monitoring and management tools are used by military organizations to provide deeper insight into the current and previous state of a mission-critical data networks for both experienced and inexperienced network administrators [33]. Those tools collect and analyze events for future analysis from a single perspective of the network: the location of the tool. The perspective of any network monitoring tool is different than the perspectives users have at the edge of the network, therefore a method is needed to analyze and depict errors from and to any point on the network.

The problem of monitoring complex networks is not new [33] [38] [19] [50], but solutions for automatically identifying complex routing errors in traditional networks (as opposed to a Software Defined Networks (SDNs)) have yet to be developed for inexperienced network administrators. Merging the concepts of defining a network architecture, collecting route forwarding state information from a live network, and visualizing the difference between the two helps less experienced network administrators troubleshoot and repair complex routing errors in mission-critical military data networks more quickly [33]. Factors such as cost, complexity, and accuracy are important in considering the efficacy of a solution that merges those concepts and were outside the scope of this research.

1.1 Military Networks

The research focused on mission-critical, military data networks and it is important to explain some constraints placed on network administrators of military networks. The constraints outlined in this section are decentralized command and control, transmission mediums, and equipment availability.

Decentralized command and control in the Marine Corps is described in Marine Corps Doctrinal Publication (MCDP) 1 [46], titled "Warfighting." MCDP 1 identifies the absolute need for decentralized command and control in the following way:

> in order to generate the tempo of operations we desire and to best cope with the uncertainty, disorder, and fluidity of combat, command and control must be decentralized. That is, subordinate commanders must make decisions on their own initiative, based on their understanding of their senior's intent, rather than passing information up the chain of command and waiting for the decision to be passed down [46, p. 78].

It is within this organizational framework that military networks are managed. Each decentralized organization has its own mission, network administrators, and equipment to operate. All of the organizations must work in harmony to establish a functional network that facilitates the communication between the organizations. MCDP 1 also addresses the nature of multiple organizations, called "elements," with differing missions working in harmony to accomplish a larger mission in the quote below:

> Each element is part of a larger whole and must cooperate with other elements for the accomplishment of the common goal. At the same time, each has its own mission and must adapt to its own situation. Each must deal with friction, uncertainty, and disorder at its own level, and each may create friction, uncertainty, and disorder for others, friendly as well as enemy.
>
> As a result, war is not governed by the actions or decisions of a single individual in any one place but emerges from the collective behavior of all the individual parts in the system interacting locally in response to local conditions and incomplete information. A military action is not the monolithic execution of a single decision by a single entity but necessarily involves near-countless independent but interrelated decisions and actions being taken simultaneously throughout the organization. Efforts to fully centralize military operations and to exert complete control by a single decision maker are inconsistent with the intrinsically complex and distributed nature of war [46, p. 12–13].

To help illustrate how decentralized command and control impacts the management of military networks, this section defines a specific example architecture that was used as the basis for this research. Although there is an infinite number of network architecture possibilities to articulate this constraint and evaluate this research against, a single example was selected to represent a common military data network and highlight important design considerations.

Figure 1.1 depicts multiple physical locations within rectangular boxes. Each physical location could be anywhere from a few hundred feet to hundreds of miles away from each other. The four locations in this example are: Main Site and Remote Sites A, B, and C. The cloud icon represents a backbone network operated by the Defense Information Systems Agency (DISA). DISA acts as an Internet Service Provider (ISP) for agencies within the Department of Defense (DoD) [23] [34], enabling satellite, data, voice, and video services throughout the world. The location labeled Main Site acts as a hub for all of the remote locations labeled Remote Site A, B, or C to access external networks, including the Internet. Redundancy across multiple gateway locations was outside the scope of this research.

Management responsibilities for the example architecture would be spread out across each site. One or more organizations would be physically located at Main Site and be responsible for

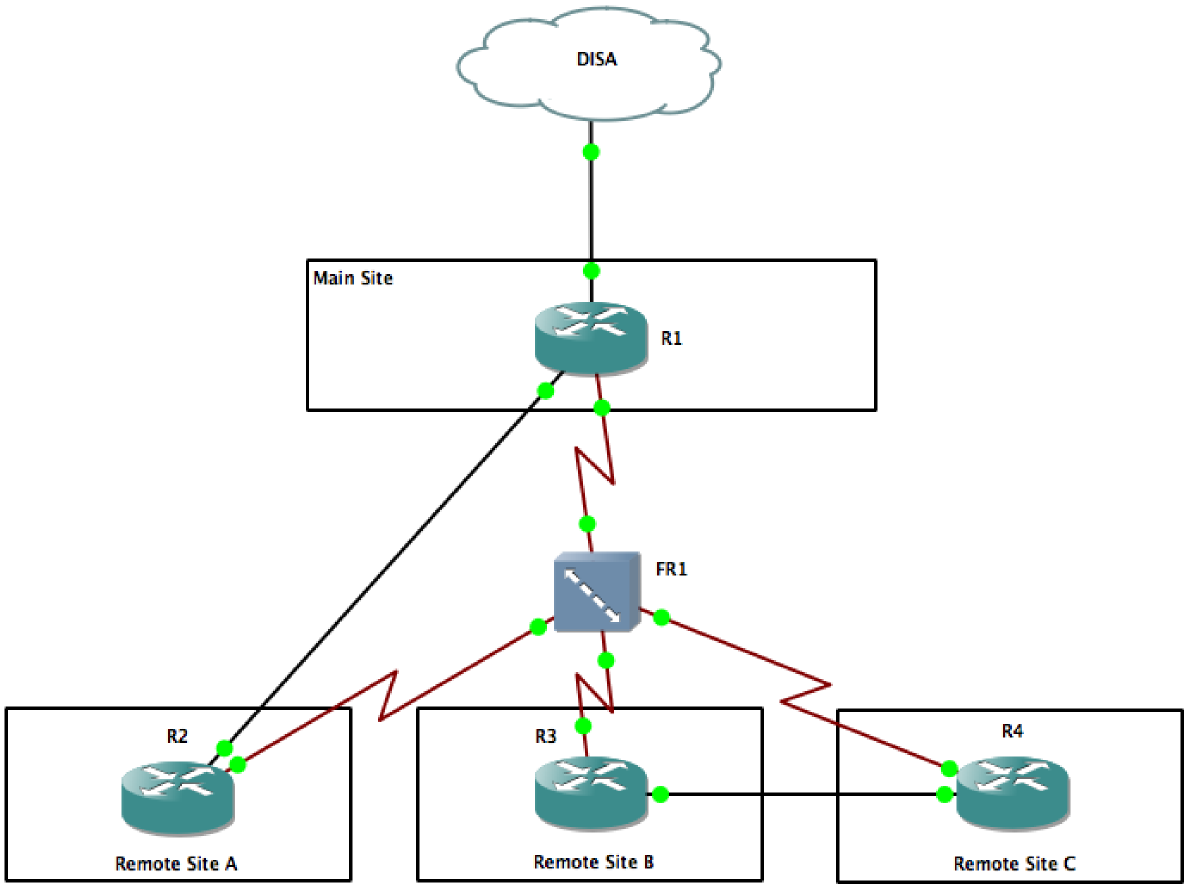


Figure 1.1: Example Architecture Diagram

managing the local network as well as the gateway connection to DISA. In this example, Main Site is also considered more important than the remote sites and subsequently would be staffed with more experienced network administrators. Remote Sites A, B, and C might have inexperienced administrators and would rely on expertise from Main Site. Decentralizing these network management functions directly correlates with the decentralized authority outlined in MCDP 1. This decentralization, while necessary for reasons outside the scope of this research, can increase the likelihood of configuration errors or mismanagement of the network [33].

The second constraint that is important to understand about military networks is why certain transmission mediums are used over others. Since the geographic area between physical locations of military units is often not controlled by friendly forces, the use of fiber between locations is not common. This generally limits the maximum bandwidth and minimum latency metrics to the capabilities and limitations of terrestrial and satellite radio systems. Terrestrial links operate at high bandwidth and low latency, when compared to satellite links, but generally require line-of-sight connectivity which can limit the range to about tens of miles. The point-to-point connections in Figure 1.1 represent terrestrial radio links between sites. Satellite links operate at low bandwidth and high latency, when compared to terrestrial links, and are not limited by the distance between locations. The point-to-multipoint connection in Figure 1.1 represents a hub-spoke topology over a Time Division Multiple Access (TDMA) satellite connection.

The last constraint to understand about military networks is the direct relationship between a network architecture and equipment availability. When deploying to a foreign country, for example, a military unit will plan an architecture based on the equipment they will take with them. Typically, there is no opportunity to purchase new equipment between the time the architecture is planned and the unit leaves. Since the architecture is predetermined and directly related to the equipment involved, the shortest path between locations is often a forgone conclusion simply based on the transmission equipment or medium used. For example, the shortest path from Remote Site A to Main Site in Figure 1.1 uses the high bandwidth, low latency, terrestrial link as the primary path. The low bandwidth, high latency, satellite link is the secondary path and is only used when the primary path is inoperable. Because of the disparity in bandwidth and latency between the primary and secondary paths, they would not be considered equal cost paths.

1.2 Research Question

This research answered the question: can route forwarding state information from individual routers be combined into a detailed visual representation of the network topology and verified against a network architecture definition in order to aid inexperienced network administrators in identifying routing errors?

This was answered in the following sections: Related Work, Architecture Definition, Network Model, and Correctness. The Related Work section analyzes literature and technical solutions to problems to this research. In the Architecture Definition section, the researcher explains how a specification was generated to define a network architecture. The next section illustrates how an architecture definition was used to define a network model. Lastly, the Correctness section describes how the researcher compared the network model against the live network from Figure 1.1.

Chapter 2

Related Work

This section explores existing, related work that was important when considering the research question. All of this work was used as a foundation to develop the tools outlined in follow-on sections.

2.1 Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT)

MAVNATT is divided into three modules: mapping, awareness, and virtualization. Combined, these three modules enabled an inexperienced network administrator to load a planned network file, compare it with a live network, and visualize whether or not the two configurations were the same.

The prototype developed during the research for MAVNATT compared a mapped topology with a virtual network by conducting a connectivity test [33, p. 58]. The MAVNATT research was focused on mapping and visualizing connectivity differences between nodes, whereas this research is focused on route forwarding state differences.

2.2 Automated Network Mapping and Topology Verification

A year after the research was completed for MAVNATT [33], another researcher continued the research by implementing the "mapping" component [21, p. 2]. The new component took a network plan as input, generated a network graph [41], and produced a visualization of the network.

Lastly, the research included a method to scan a live network to determine if the live network was the same as the model network or not. Like the previous research, this research also focused on connectivity and node state from the perspective of the tool itself rather than focusing on route forwarding state between the nodes.

2.3 Ordinal Process Network Evaluation Tool (OPNET) Modeler

Alain Cohen's research on "Simulating Virtual Circuits in Mobile Packet Radio Networks" [20] at the Massachusetts Institute of Technology (MIT) in 1986 lead to the development of a product used by many public and private entities called Ordinal Process Network Evaluation Tool [12] (OPNET) Modeler. The resulting product is currently used by the DoD, although it was re-branded as the Joint Communication Simulation System [7] (JCSS) and contains additional network equipment models unique to the DoD. JCSS is used to validate network architecture plans and analyze the performance of new devices and applications. This is useful during the acquisition of equipment and planning for large data networks. Because of the learning curve of both the software and its underlying principles, however, it is not suitable for use by inexperienced network administrators [25, p. 8].

The focus of OPNET was to be a comprehensive modeling product [25], whereas the focus of this research is defining an architecture, modeling that architecture, and determining the correctness of a live network for inexperienced administrators. The model developed in this research contained enough connectivity information to leverage existing graph theory tools in order to determine shortest path information between nodes and evaluate correct route flows for comparison against a live network.

2.4 Route Explorer

Route Explorer is a commercial product made by Packet Design, Inc [38]. It provides a view of the entire routing topology of a live network. This tool enables visualization of the routing topology by "monitoring the routing protocols that direct the flow of traffic throughout the network, Route

Explorer constructs the routers' view of the network, computing and displaying topology changes and routes in real time" [5, p. 2]. This research expanded on the concepts in Route Explorer's live view of the network by first defining an architecture and then validating route forwarding state correctness based on that architecture.

2.5 Thousand Eyes

The network monitoring solution produced by Thousand Eyes, Inc [30] [1] is a commercial product that provides intelligent, clear visualization of a live network. Additionally, it provides information to a network administrator regarding the availability of nodes along a given path. This tool does not attempt to validate route forwarding state correctness based on a previously defined network architecture. As with Route Explorer, this research expanded on the concepts in Thousand Eyes' view of the network by first defining an architecture and then validating route forwarding state correctness based on that architecture.

2.6 Internet Protocol (IP) Address Management (IPAM)

Internet Protocol (IP) Address Management (IPAM) is used commonly throughout the DoD to help identify and manage IP address allocation. IPAM tools, such as SolarWinds IP Address Manager [10], provide insight into the network by maintaining state information about IP address allocation assignment to specific subnets. That information is extended by allowing the administrator to combine subnets into a physical site or location. Mapping subnets to physical sites does not provide enough information to build a model representation of the network because it lacks critical information about network connectivity, such as bandwidth and delay of a link between routers.

IPAM tools help identify endpoints on a network and those endpoints can provide visibility into the internal network. Using end-point data to determine internal characteristics of a network is called Network Tomography [47] [31]. This process is useful when analyzing a network that is not administrated by the analyzer because elevated network access privileges are not required.

However, more information can be gathered by inspecting network elements directly using elevated network access privileges.

2.7 Graph Markup Language (GraphML)

GraphML [15] is a file format based on XML [45] meant to "standardize [graph representations] for the graph drawing community" [6]. It is a standard schema for graphs, nodes, edges, and generic "data." Implementations can use the generic "data" node to represent any combination of key and value required.

Related research [33] [21] developed a prototype using GraphML to represent a network architecture. It was determined that both GraphML and JavaScript Object Notation (JSON) met the research goals because they were "simplistic and lightweight solutions that will allow for faster prototyping and deployment" [33, p. 44].

GraphML does not provide a schema for representing node configuration data or operational state. In order to differentiate whether a node represents a switch or router, the "GraphML Network Topology" XML file in the research by McBride [33, fig. 26] establishes a key "d2" called "device" of type "string." While GraphML is lightweight and simple, it lacks specificity for this use. Implementing a standard, comprehensive, and consistent schema to represent data network configurations properly would not take advantage of any existing standards available throughout the network configuration community. Additionally, GraphML is focused on representing relationships between nodes and edges and not all information rely on understanding these relationships. Inspecting or validating bandwidth of a single interface on a node is irrelevant to the node's relationship with the rest of the network.

2.8 YANG and OpenConfig

In 2010, the Internet Engineering Task Force (IETF) published a standard for representing configuration and state data for the Network Configuration (NETCONF) Protocol [39] called YANG. YANG is defined as:

> a language used to model data for the NETCONF protocol. A YANG module defines a hierarchy of data that can be used for NETCONF-based operations, including configuration, state data, Remote Procedure Calls (RPCs), and notifications. This allows a complete description of all data sent between a NETCONF client and server [32, ch. 4.1].

SDN administrators are already familiar with NETCONF and YANG because that is one method of communication between SDN components as well as external network monitoring and management systems [27]. Using that protocol to collect information from traditional networks cannot be relied upon for this research because traditional data networking equipment might not support NETCONF and the federal government primarily uses traditional data networking equipment. According to a recent survey, only 37% of federal government organizations currently use any SDN capabilities [22, p. 3].

One effort to standardize the decoupling of YANG from NETCONF transport protocol is called OpenConfig. OpenConfig is defined as:

> Based on this operational requirement, this document seeks to enumerate the requirements of representing both configuration and operational state data in YANG; propose a common set of terminology; and propose a common layout for configuration and state data such that they can be retrieved from a [network element]. These proposals are based on the assertion that YANG models should be usable via a number of protocols (not solely IETF- defined protocols such as NETCONF and RESTCONF), and may also be used to carry data that is pushed from devices via streaming rather than polled [43].

Since OpenConfig aims to be vendor-agnostic, is decoupled from transport protocols, and can be used to represent both traditional and SDN components, it is a good candidate to be used as a foundation in the architecture definition section of this research.

Chapter 3

Architecture Definition

Marine Corps data network architectures are currently represented by static diagrams that are drawn and reviewed by various organizations prior to being given the approval to operate on the Defense Information Systems Network (DISN) [4, ch. 3.8]. Rarely do the collection of static diagrams contain enough information for inexperienced administrators to install and operate complex military data networks. Instead of static diagrams, the architecture can be represented by an Architecture Definition Language (ADL) that describes the configuration and operational state data within the architecture–such as router nodes, links between those nodes, transmission mediums, bandwidth, delay, and route flow information.

The ADL is used to build a model of the planned architecture. This might, for example, specify that a default route originates from a specific node or that traffic should prefer a low latency primary transmission path over a high latency backup transmission path. The ADL must be able to define the physical and data-link network topologies of the network along with route flows. Most DoD networks, for example, have one or more default routes to the DISA core network in order to reach other units and agencies within the United States government or even multinational partners. Representing which physical path should be primary or alternate for the default route is required to accurately represent complex military data network architectures.

The research found no existing, widespread standard for representing the required elements of an architecture definition in traditional networks. Although there is significant interest in network architecture documentation, as seen through Cisco's addition of Design and Architecture

Certifications [2] to their list of certifications for network professionals, no formal solution for programmatically representing a network architecture has been adopted widely among traditional, or non-SDN, equipment. Command Line Interface (CLI) and Simple Network Management Protocol [18] (SNMP) are the two most common ways to access network elements, however, they are not vendor-agnostic nor are they comprehensive enough to meet the requirements defined in this research.

As discussed in section 2.8, OpenConfig is an emerging standard that looks promising as an ADL. It is gaining industry support and aims to be vendor agnostic. Because this is still an emerging standard and conflicts with the YANG standards published by the IETF, it was not suitable for use in this research. In the interim, the research focused on defining enough information within an OpenConfig-like ADL to build a model of the entire architecture within a single application.

In theory, the definition language should be broad enough to handle all types of network architecture requirements. In practice, however, the definition was limited in scope to address the research question and scope. The ADL used in Figure 3.1 satisfied the research scope of defining a correct architecture definition for the purposes of building a correct architecture model.

```
{
  "devices": [
    {
      "name": "R1",
      "mgmt_address": "192.168.255.1",
      "interfaces": {
        "fa0/0": {
          "ip_addresses": [
            "192.168.56.103/24"
          ],
          "bandwidth": "100 Mb",
          "delay": 1,
          "cost": 1
        },
        "fa1/0": {
          "ip_addresses": [
            "10.0.2.1/30"
          ],
          "bandwidth": "100 Mb",
          "delay": 1,
          "cost": 1
        },
        "s3/0.1": {
          "ip_addresses": [
            "192.168.0.1/24"
          ],
          "bandwidth": "1.544 Mb",
          "delay": 10,
          "cost": 2
        },
        "l0/0": {
          "ip_addresses": [
            "192.168.255.1/32"
          ],
          "cost": 0
        }
      }
    },
    {
      "name": "R2",
      "mgmt_address": "192.168.255.2",
      "interfaces": {
        "fa0/0": {
          "ip_addresses": [
            "10.0.2.2/30"
          ],
          "bandwidth": "100 Mb",
          "delay": 1,
          "cost": 1
        },
        "fa1/0": {
          "ip_addresses": [
            "10.2.0.1/24"
          ],
          "bandwidth": "1 Gb",
          "delay": 1,
          "cost": 1
        },
        "s3/0.1": {
          "ip_addresses": [
            "192.168.0.2/24"
          ],
          "bandwidth": "1.544 Mb",
          "delay": 10,
          "cost": 2
        },
        "l0/0": {
          "ip_addresses": [
            "192.168.255.2/32"
          ],
          "cost": 0
        }
      }
    },
    {
      "name": "R3",
      "mgmt_address": "192.168.255.3",
      "interfaces": {
        "fa0/0": {
          "ip_addresses": [
            "10.0.1.2/30"
          ],
          "bandwidth": "100 Mb",
          "delay": 1,
          "cost": 1
        },
        "fa1/0": {
          "ip_addresses": [
            "10.3.0.1/24"
          ],
          "bandwidth": "1 Gb",
          "delay": 1,
          "cost": 1
        },
        "s3/0.1": {
          "ip_addresses": [
            "192.168.0.3/24"
          ],
          "bandwidth": "1.544 Mb",
          "delay": 10,
          "cost": 2
        },
        "l0/0": {
          "ip_addresses": [
            "192.168.255.3/32"
          ],
          "cost": 0
        }
      }
    },
    {
      "name": "R4",
      "mgmt_address": "192.168.255.4",
      "interfaces": {
        "fa0/0": {
          "ip_addresses": [
            "10.0.1.1/30"
          ],
          "bandwidth": "100 Mb",
          "delay": 1,
          "cost": 1
        },
        "fa1/0": {
          "ip_addresses": [
            "10.4.0.1/24"
          ],
          "bandwidth": "1 Gb",
            "delay": 1,
            "cost": 1
          },
          "s3/0.1": {
            "ip_addresses": [
              "192.168.0.4/24"
            ],
            "bandwidth": "1.544 Mb",
            "delay": 10,
            "cost": 2
          },
          "lo/0": {
            "ip_addresses": [
              "192.168.255.4/32"
            ],
            "cost": 0
          }
        }
      },
      {
        "name": "DISA",
        "interfaces": {
          "fa0/0": {
            "ip_addresses": [
              "192.168.56.1/24"
            ],
            "bandwidth": "100 Mb",
            "delay": 1,
            "cost": 1
          }
        }
      }
    ],
    "locations": [
      {
        "name": "Main Site",
        "coordinates": [ 100, 100 ],
        "size": [ 800, 300 ],
        "devices": [
          "R1",
          "10.1.0.0/24"
        ]
      },
      {
        "name": "Remote Site A",
        "coordinates": [ 100, 500 ],
        "size": [ 200, 200 ],
        "devices": [
          "R2",
          "10.2.0.0/24"
        ]
      },
      {
          "name": "Remote Site B",
          "coordinates": [ 400, 500 ],
          "size": [ 200, 200 ],
          "devices": [
            "R3",
            "10.3.0.0/24"
          ]
        },
        {
          "name": "Remote Site C",
          "coordinates": [ 700, 500 ],
          "size": [ 200, 200 ],
          "devices": [
            "R4",
            "10.4.0.0/24"
          ]
        }
      ]
    }
```

Figure 3.1: Sample Architecture Definition Language

Chapter 4

Network Model

Using the ADL defined in chapter 3, the researcher created a model to represent the network architecture. The model, a graph simulation of all nodes and edges, was built using the NetworkX Python software library. NetworkX is a:

> Python language package for exploration and analysis of networks and network algorithms. The core package provides data structures for representing many types of networks, or graphs, including simple graphs, directed graphs, and graphs with parallel edges and self-loops [42].

NetworkX is used to build networks as defined in the research area of graph theory [41]. In that lexicon, a data network router is called a “node” or “vertex” and a transmission link (the connection between routers) is called an “edge” [48] [41]. For the purposes of this research, the network of nodes and edges was created using a directed graph with self-loops and parallel edges. This type of graph, known as a multi-digraph, allowed nodes to connect to themselves and other nodes multiple times. The multi-digraph most closely resembled a network of routers and transmission links. Building a multi-digraph of all nodes and edges in the architecture definition allowed the researcher to evaluate the shortest path between nodes using appropriate cost metrics without querying live devices.

4.1 Hypergraph

The multi-digraph did not support a concept used in the Example Architecture (Figure 1.1): a point-to-multipoint data network. This concept is most closely defined in graph theory as a “hypergraph” [26] [52] [28]. In general terms, a hypergraph is a set of nodes that share a common edge [52]. The NetworkX software library does not support hypergraph natively. The closely-related bipartite graph [52] is supported by NetworkX, but cannot be used with shortest path algorithms. Through this research and related work [40, ch. 4.3], it was determined that a multi-digraph is the simplest way to represent the overall Example Architecture (Figure 1.1) and the point-to-multipoint hypergraph could be translated into an equivalent representation in the multi-digraph.

The point-to-multipoint data network in the Example Architecture (Figure 1.1) was converted from a hypergraph to a multi-digraph using the technique described by Ritz and Murali [40, ch. 4.3]. Figure 3 in the research by Ritz and Murali [40, fig. 3] visualizes the technique that satisfies the constraints of a multi-digraph: an edge connects exactly one or two nodes. Figure 4.1 is resulting multi-digraph and showed that the visualization of the network did not match the point-to-multipoint nature of the network–it changed the relationship from point-to-multipoint to point-to-point.

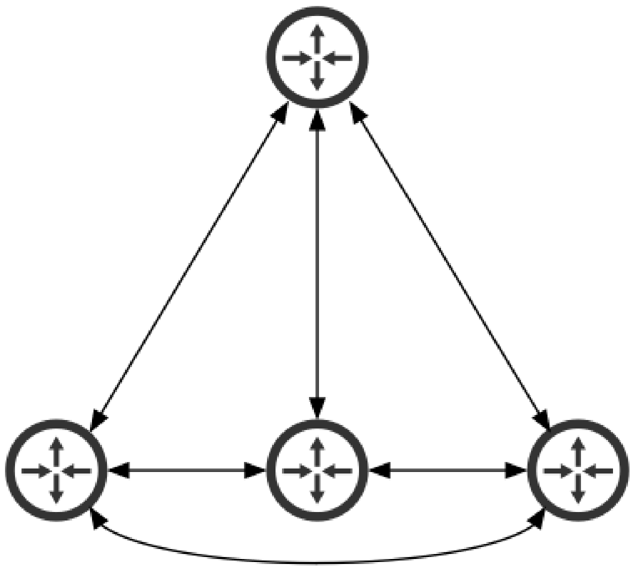


Figure 4.1: Hypergraph as a Multi-Digraph

To create a multi-digraph that could be visualized in the same manner as it was analyzed as a model, the researcher created an artificial node between router nodes to represent a multiple access network. This was similar to Figure 2.8 in “Visualization of Hyperedges in Fixed Graph

Layouts" [28]. Figure 4.2 and Figure 4.3 are visualizations of the point-to-multipoint network before and after adding a special node between the router nodes, representing the multiple access nature of the network.

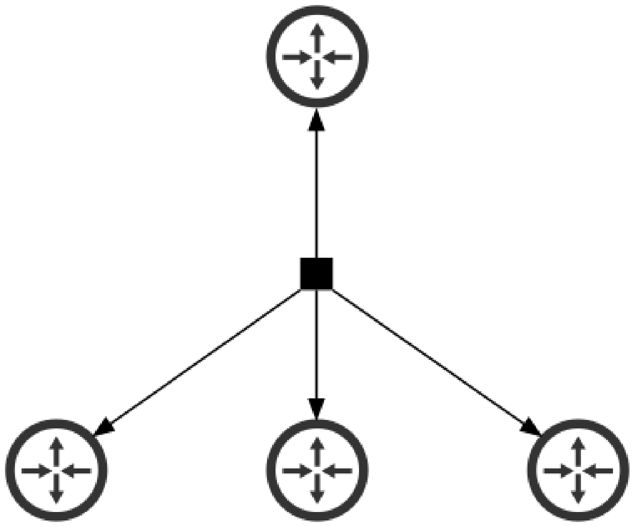


Figure 4.2: Hypergraph Visualization in Fixed Graph Layouts

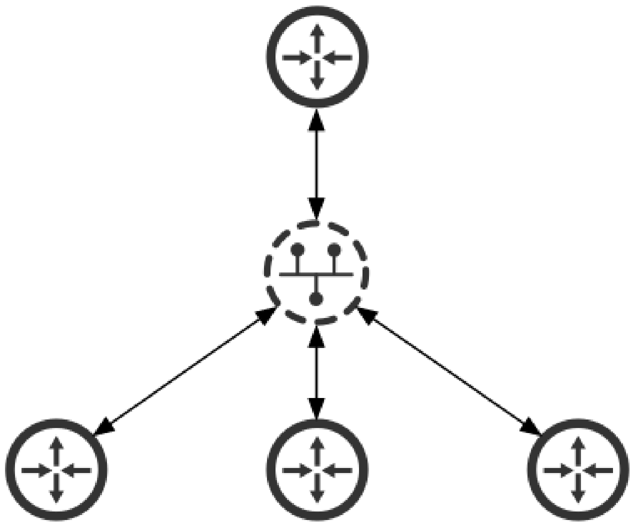


Figure 4.3: Hypergraph Visualization with Subnet Icon

Ensuring that shortest path calculations were not altered by the presence of this special node, a special node was created for every subnetwork with two or more connected devices–even point-to-point networks. Additionally, to not alter the cost metric calculation of the shortest path algorithm, the directional edge connecting the router node and the virtual subnetwork node contained all of the bandwidth, delay, and cost metric attributes. All directional edges connecting the virtual subnetwork node to router nodes contained zero values for bandwidth, delay, and cost. Figure 4.4 visually depicts where the metrics were applied for a point-to-multipoint network.

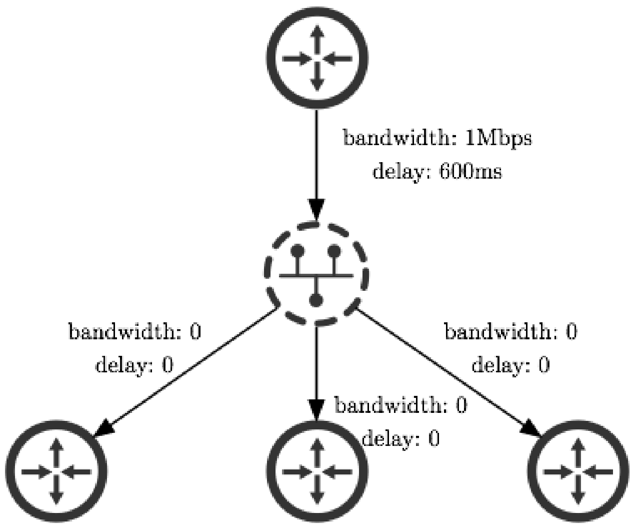


Figure 4.4: Hypergraph Visualization with Appropriate Metrics

4.2 Model Implementation

The model implementation consisted of NetworkX node and edge instances for every router and transmission link in the Example Architecture (Figure 1.1). The relationship between those nodes and edges was examined using graph theory concepts [41] implemented in the NetworkX software library. Unlike OPNET modeler [12] and JCSS [7], the model in this research did not implement any routing protocols to share state between nodes. Defining nodes and edges with appropriate attributes such as bandwidth and delay, enabled the functions in NetworkX to determine the shortest path between nodes.

The model used Dijkstra's Shortest Path First (SPF) algorithm [24] to match the Open Shortest Path First (OSPF) protocol algorithm [36] used in the Example Architecture (Figure 1.1). On Cisco devices, the calculated cost of an OSPF interface is $\frac{bandwidth_{ref}}{bandwidth_{int}}$ where $bandwidth_{ref}$ is

100Mbps and $bandwidth_{int}$ is the interface bandwidth across a specific transmission medium.

In military data networks, it is common for the router interface bandwidth to be greater than the maximum bandwidth available over the transmission equipment. For example, TDMA satellite modems connect to routers over 100Mbps Fast Ethernet and transmit data over a satellite channel with less than 15Mbps of bandwidth shared between all channel subscribers. If the maximum bandwidth over the transmission equipment is not specified on the router interface, either the live network or the model would incorrectly choose this low bandwidth link over a different, high bandwidth link because the transmission bandwidth didn't match the physical interface bandwidth.

4.3 Visualizing the Model

Visualizing the architected network in order to aid inexperienced network administrators in identifying routing errors was a foundational aspect of this research because visualization "increases our ability to perform [discovery, decision making, and explanation] and other cognitive activities" [17, p. 6]. Using an ADL to accurately model the state of a complex data network was the first step. Next, a dynamic visualization of the model was built as a foundation for determining the correctness of the network in the next section.

Figure 4.5 is a dynamic visualization of the network model, created using a Python server and a web browser front-end application. The Python server read the ADL input, built the NetworkX model, and started a web server as a method for the front-end application to access the NetworkX model. The front-end application was built using Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and Javascript. The visualization component used in the front-end was a Javascript library called Data-Driven Documents (D3) [14]. D3 was created at the Stanford Visualization Laboratory in 2011 as a successor to Protovis [13], in order to present a "novel representation-transparent approach to visualization for the web" [14, p. 1].

Specifically, the visualization of the network model used D3's force-directed layout animation functions to represent nodes and their connectedness via edges and edge attributes. The visualization developed for this research focused on representing routers and subnets as "nodes" and

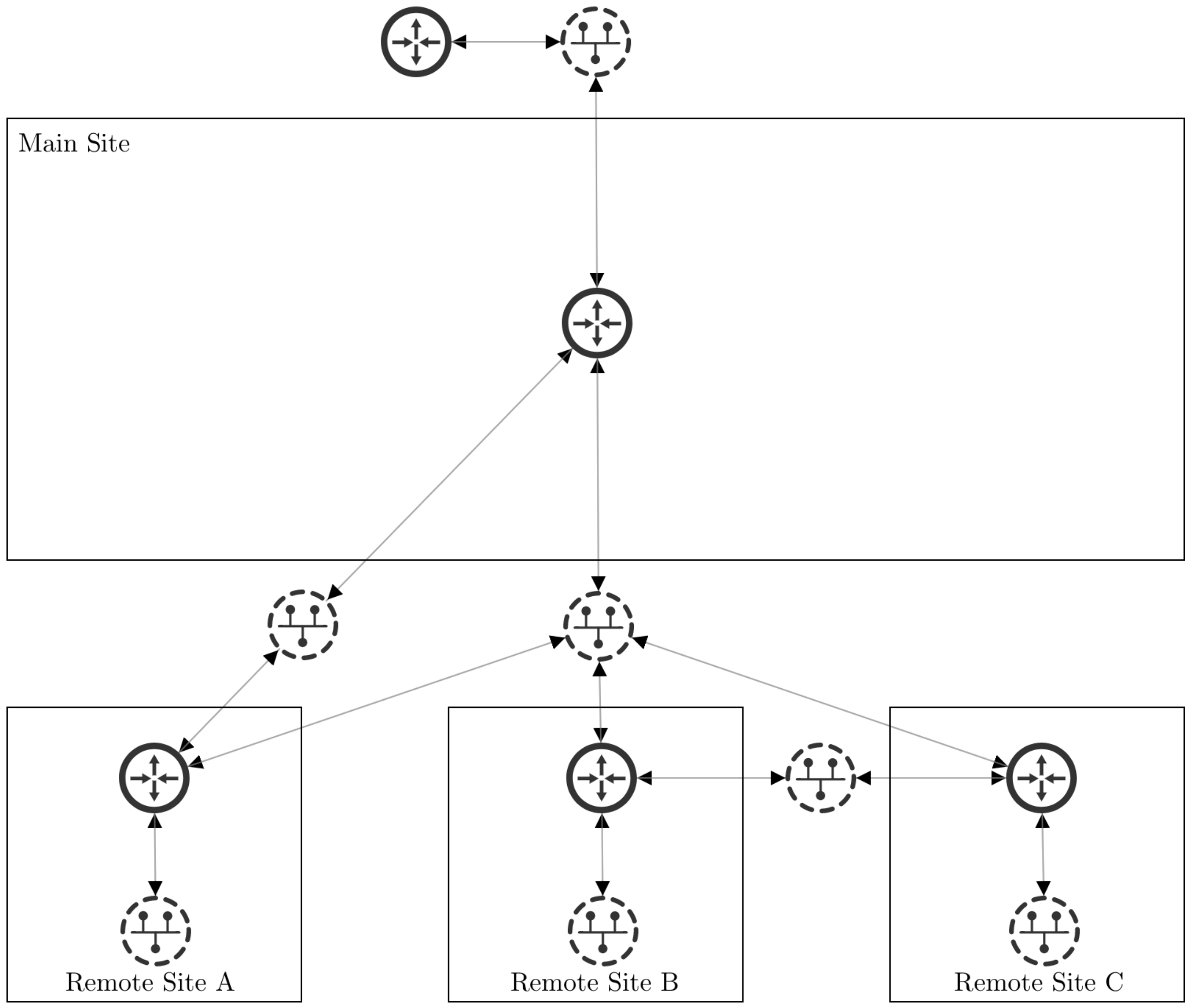
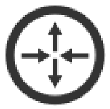


Figure 4.5: Dynamic Model Visualization

network-layer connectivity as "edges." As mentioned in the Military Networks section (section 1.1), physical locations are critical when depicting complex military data networks because they often represent the boundary between friendly- and enemy-controlled areas. The architecture definition outlined locations and the nodes contained therein. Locations were represented by rectangular boundaries. Devices that primarily operate at the network layer were represented by Figure 4.6a. Subnets were represented as virtual nodes by Figure 4.6b.


(a) Router Icon

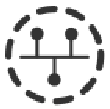

(b) Subnet Icon

Figure 4.6: Network Element Icons

Combining D3 with standardized representations of network elements such as locations, devices, and connections enabled the research tool to accurately depict the correct network model to an inexperienced administrator. Additionally, it was the foundation for both collecting live network information and visualizing the shortest path between networks.

Chapter 5

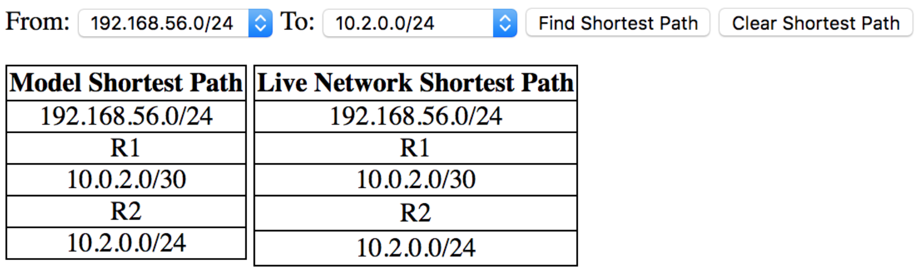
Correctness

An inexperienced network administrator can use the correct architecture model and the visualization of that model as a foundation for displaying route forwarding state information. Because the visualization is already at a known good state, errors in the live network, such as link failures or configuration issues, can easily be overlaid on to the visualization to highlight the differences between the model and the live network.

Additionally, the model can be used as a starting point for inspecting the live network. Various network management tools use the concept of a seed router to begin inspecting certain aspects of a network. For example, a network management tool that identifies router nodes on a network might begin by inspecting a given seed router for all of its neighbor addresses. That information can be combined into a larger set of known devices on the network. The model defined in this research was used as a seed router, obviating the need to specify a seed router.

5.1 Shortest Path

The researcher determined the correctness of the live network by comparing the shortest path between two subnetworks on the model with the shortest path between those same two subnetworks on the live architecture. Figure 5.1 shows a form in the front-end application for a user to identify a source and target subnetwork. When a user submited the form, the front-end application sent the source and target information to the server which performed the shortest path lookup on both the model and the live network.



Model Shortest Path
192.168.56.0/24
R1
10.0.2.0/30
R2
10.2.0.0/24

Live Network Shortest Path
192.168.56.0/24
R1
10.0.2.0/30
R2
10.2.0.0/24

Figure 5.1: Shortest Path Form

Network management tools like ping [29, ch. 3.2] and traceroute [29, ch. 3.4] require the network administrator to specify a target node or network to determine the path to. Those tools use the node they are executed on as the source and the path to the target node is from that perspective. The tool developed to answer this research question instead allowed the administrator to identify both the source and target networks. This meant that the network could be viewed from the perspective of any node to any other node–it was not limited to a single perspective.

After selecting the source and target networks, the model was inspected to determine the shortest path between the identified networks. It is important to note that the OSPF protocol was used on the live network as the Interior Gateway Protocol (IGP). In order to ensure parity between shortest path calculations on the live network and the model architecture, the NetworkX `dijkstra_path()` function was used which is an implementation of Dijkstra's SPF algorithm [24]. Figure 5.2 is the source code depicting two important functions that were developed as part of this research: the interface cost calculation function and the shortest path function.

```python
class OSPFTopology(Topology):
    reference_cost_bandwidth = bitmath.Mb(100)

    def calculate_interface_cost(self, node: Layer3, iface: Interface):
        if iface.ospf_cost is not None:
            cost = iface.ospf_cost
        else:
            cost = OSPFTopology.reference_cost_bandwidth / iface.bandwidth

        return cost
```

```
def calculate_shortest_path(self, source: Union[Layer3, Subnet],
    target: Union[Layer3, Subnet]):
    try:
        path = nx.dijkstra_path(self.G, source, target, weight='cost')
    except nx.NetworkXNoPath:
        path = None

    return path
```

Figure 5.2: Model OSPF Cost and Shortest Path Functions

After the shortest path was calculated on the model, the first router in the shortest path was used as the starting router to inspect the live network.

5.2 Collection

There are two vendor-agnostic, defacto standard ways to collect route forwarding state information from traditional network equipment remotely: SNMP [18] and Secure Shell [53] (SSH). Both protocols are either encrypted by default or can be encrypted and thus are acceptable means of remotely connecting to hardened DoD data network infrastructure, according to DISA's Standard Technical Implementation Guides [8] (STIGs).

This research used SSH to connect to remote, virtual Cisco routers using a Python library called netdev [51]. This library used SSH to connect to the router's Command Line Interface (CLI) and execute commands. After the command is executed, the resulting output was captured by the Python library and returned to the caller. In this case, the researcher utilized TextFSM, a "Python module for parsing semi-structured text into Python tables" [11] and existing templates from the Network to Code (NTC) open source library [9] to parse the `show ip route` command results into a table. That table was then entered into a virtual instance of the router node within the architecture model as an in-memory copy of the live routing table. Another Python library, called PyTricia [44], was used to store the routes in a longest prefix match table for quick, router-like querying of the "best" route [35] [49]. Figure 5.3 is the code segment that determined the shortest path between two subnetworks on a live network.

```
async def shortest_path(self, source, target, model_shortest_path, existing_path=None):
    all_paths = []
    node_stack = []

    if existing_path is None:
        # we're starting at the beginning
        path = []

        # here we can use a shortcut to identify the first router in the live network
        # that we can access by mgmt_address using information from the model's
        # shortest path.
        for node in model_shortest_path:
            path.append(node)

            if isinstance(node, Layer3) and node.mgmt_address is not None:
                # Find the first real router and add it to the stack.
                node_stack.append(node)
                self.logger.debug("starting live node is: %s" % node)
                break
    else:
        path = existing_path
        # the last item in the path is the router we need to start with
        node_stack.append(path[-1])

    # at this point, we have a path with the source and the first router in it.
    # Everything from here is based on real route tables
    while len(node_stack) > 0:
        current_node = node_stack.pop()
        await current_node.get_route_table()

        routes = current_node.route_table.get(target)
        if routes is None:
            # if we don't find a route, break out of the while loop and return
            break

        path_until_routes = path.copy()
        for index, route in enumerate(routes):
            next_hop_dict = self.nodes_by_interfaces.get(route.next_hop_ip)
            if index == 0:
                if route.origin_protocol == 'connected':

                    if isinstance(current_node, Layer3) and \
                            await current_node.has_l3_address(target):
                        # this is the target!  Don't do anything because we've already
                        # added this node
                        pass
                    elif target == str(route.network):
                        path.append(route.network)
                    else:
                        # the route is connected, however we need to find the
                        # very last hop to terminate the shortest path
                        for n in self.topo.G:
                            if n == target:
                                path.append(n)
                            else:
                                if isinstance(n, Layer3) and \
                                        await n.has_l3_address(target):
                                    path.append(route.network)
                                    path.append(n)
                                    break

                elif next_hop_dict is None:
                    self.errors.append('No next hop found for %s' % route.next_hop_ip)
                else:
                    next_hop = next_hop_dict['node']
                    path.append(next_hop_dict['subnet'])
                    path.append(next_hop)
                    node_stack.append(next_hop)
            else:
                next_hop = next_hop_dict['node']
                recursive_path = path_until_routes.copy()
                recursive_path.append(next_hop_dict['subnet'])
                recursive_path.append(next_hop)
                all_paths += await self.shortest_path(
                        source,
                        target,
                        model_shortest_path,
                        existing_path=recursive_path)

    all_paths.append(path)
    return all_paths
```

Figure 5.3: Shortest Path Detection on Live Network

Collecting current route forwarding state information from a small number of routers in a military data network was not computationally intensive. As the number of routers on the network increases so does the complexity in obtaining current forwarding state information in a reasonable amount of time, according to the research by Zeng [54] and Xie [50]. Unfortunately, the solutions described in the research by Zeng and Xie leverage the centralized state information from SDN, which does not exist in traditional data networks. Addressing this scaling limitation was outside the scope of this research.

After the routing table was collected from the first node in the path, the longest prefix route was selected from all available routes in PyTricia [44] for that node. The result of that query was the next hop along the shortest path. For each next hop found, the tool connected to that router using netdev and collected its route table for inspection until no further hops were found. Stepping through the network like this allowed the tool to find the best route from each node in real time.

5.3 Visualizing the Shortest Path

The shortest path between the source and target networks was highlighted on the model visualization for the network administrator. The shortest path determined by the model used the connectivity lines already drawn on the visualization and changed the line color. The shortest path determined by the live network was depicted by drawing additional, curved lines between nodes and highlighted those in a different color from everything else on the visualization. Figure 5.4 shows the Example Architecture (Figure 1.1) with the shortest path determined by the model as straight green lines and the shortest path determined by the live network as curved blue lines.

If the two shortest path calculations were the same, then the live network was operating correctly. If the two shortest path calculations were different, then there was an error on the live network that must be addressed. Figure 5.5 is an example of this mismatch. The straight green lines go from the router icon at the top of the figure to the subnet icon at the bottom of the figure. The curved blue lines, however, start and end at the router icon at the top of the figure after traversing the router at Main Site.

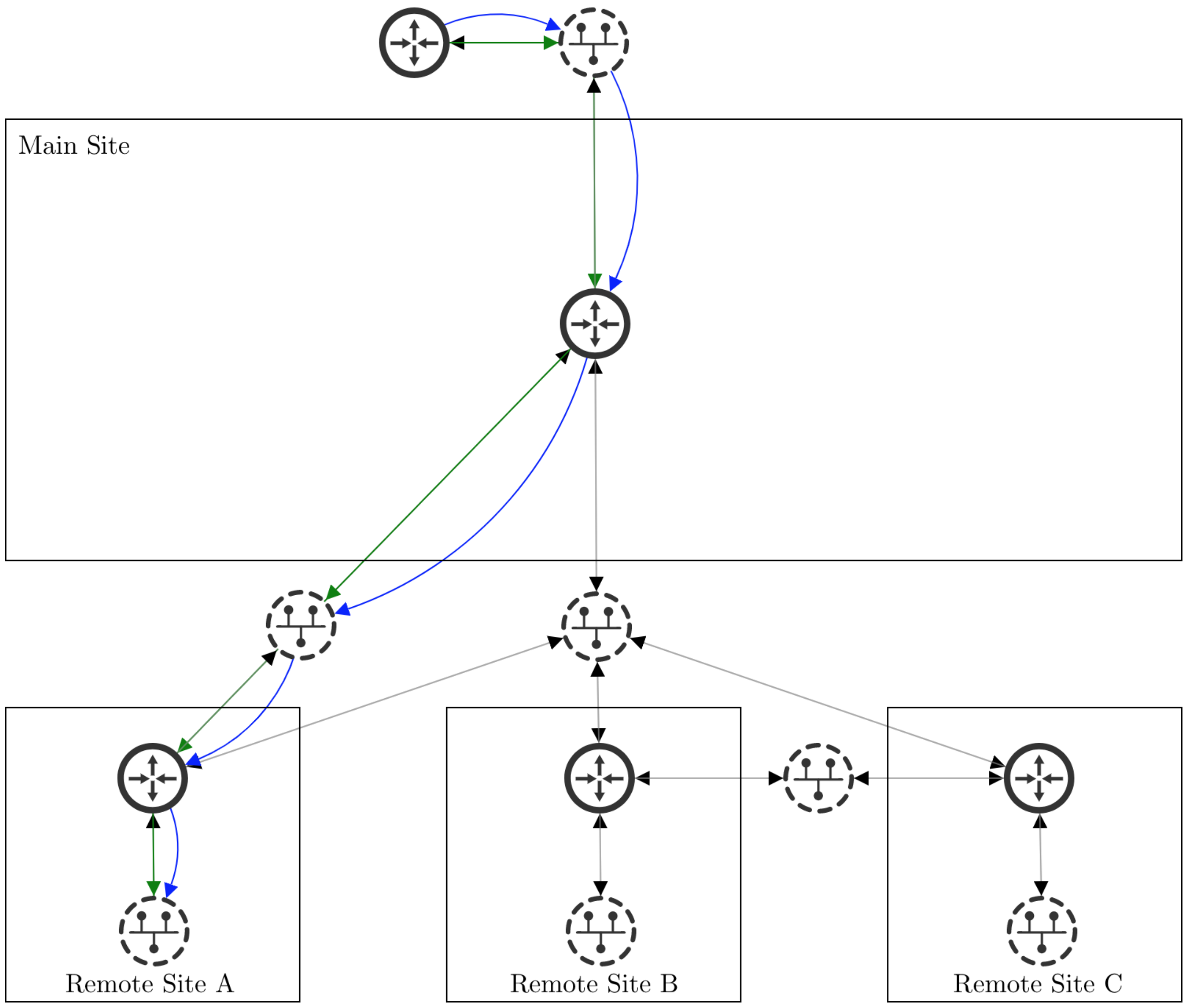


Figure 5.4: Dynamic Visualization of Matching Shortest Paths

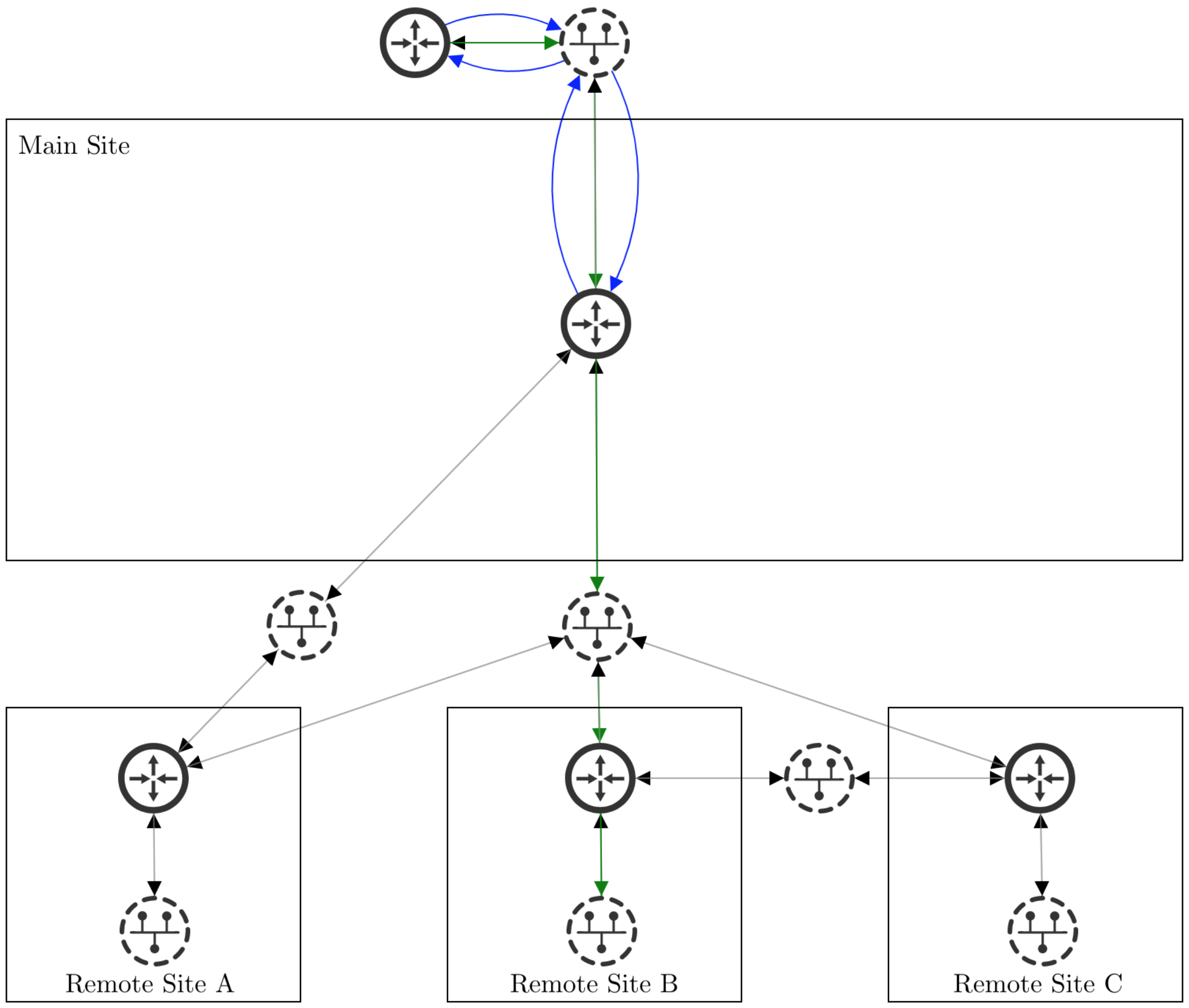


Figure 5.5: Dynamic Visualization of Mismatching Shortest Paths

Chapter 6

Conclusion

The research showed that route forwarding state information from individual routers could be combined into a detailed visual representation of the entire network topology and verified against a network architecture definition in order to aid inexperienced network administrators in identifying routing errors. The solution was broken into three areas: architecture definition, network modeling, and determining correctness.

A solution was implemented that utilized an ADL to describe a network architecture. The ADL included physical and data-link attributes that were required to build an accurate model. A Python server imported the architecture definition and created a NetworkX model of the architecture, defining relationships between all nodes and edges. A web browser connected to the Python server and displayed a visualization of the NetworkX model. Router and subnet icons were used to represent the model nodes and directional lines represented the edges between those nodes. The solution allowed a user to select source and target subnetworks and compare the shortest path between them on both the architecture model and live network. Lastly, the shortest paths determined from the model and live network were highlighted on the visualization.

The results of this research can positively impact how mission-critical, military data networks are planned, installed, operated, maintained, and defended. An experienced network architect can use an ADL to describe how the network should work and network administrators at any experience level can visualize how a live network compares to the architected network. Research that helps an inexperienced network administrator identify and resolve routing errors in mission-critical, military data networks enhances the operational effectiveness of the military.

6.1 Future Work

Further research can be conducted to improve and add additional methods for identifying routing errors using route forwarding state in mission-critical military data networks. Additional research is recommended in the following areas: implementing OpenConfig as an ADL; incorporating real-time, secure network visualization updates; and implementing additional routing protocol analysis.

6.1.1 OpenConfig as an ADL

This research determined that OpenConfig was a good candidate for an ADL because it is gaining industry support and aims to be vendor agnostic. Additional research in this area could determine the efficacy of using an emerging standard as an ADL, especially when some modules conflict with IETF standard YANG modules. If OpenConfig could be used as an ADL, it could help extend the work from the SDN community into traditional network management tools.

6.1.2 Real-Time Updates

The front-end application components developed to answer this research question allowed the user to update the network visualization by reloading the page or submitting a form to query the server for the shortest path between two elements. Further research into how the front-end and server components could support real-time updates over a secure, web-based connection would further automate the process. Additionally, it would be useful to determine the limitations of different approaches to collecting and evaluating the route forwarding state from a live network as part of a real-time application.

6.1.3 Additional Routing Protocols

OSPF was chosen as the routing protocol to evaluate this research because NetworkX supported the same algorithm at the heart of OSPF. Additional research should implement methods that determine the shortest path between nodes of a network graph using algorithms from routing protocols such as Border Gateway Protocol (BGP), Enhanced Interior Gateway Router Protocol (EIGRP), Routing Information Protocol (RIP), and Intermediate System to Intermediate System (IS-IS). This could also compare the route forwarding state from individual routing topology tables with the global routing table to help identify the location of routing errors.

Bibliography

[1] Network Device Monitoring. https://www.thousandeyes.com/solutions/network-device-monitoring.

[2] Cisco Design and Architecture Certifications, 2013.

[3] 2015 Demographics - Profile of the Military Community, 2015.

[4] Defense Information Systems Network Connection Process Guide, September 2016.

[5] Route Explorer Datasheet, 2016.

[6] About GraphML. http://graphml.graphdrawing.org/about.html, July 2017.

[7] Joint Communication Simulation System Model Library & Features. Technical report, Defense Information Systems Agency, 2017.

[8] Network Management Security Guidance At-a-Glance Version 9, Release 1, August 2017.

[9] Ntc-ansible: Multi-vendor network modules, October 2017.

[10] SolarWinds IP Address Management Software. http://www.solarwinds.com/ip-address-manager, 2017.

[11] Textfsm: Python module for parsing semi-structured text into python tables, October 2017.

[12] David M. Banker. Modeling and Simulation of Communication Systems in OPNET. PhD thesis, Air Force Institute of Technology, March 2000.

[13] Michael Bostock and Jeffrey Heer. Protovis: A Graphical Toolkit for Visualization. IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis), 2009.

[14] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-Driven Documents. IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis), 2011.

[15] Ulrik Brandes, Markus Eiglsperger, Jürgen Lerner, and Christian Pich. Graph Markup Language (GraphML). 2013.

[16] Joseph E. Buder, Amanda W. Gladney, and James B. Nazar. Retention of Computer Network System Administrators in the Air Force. Technical Report AU/ACSC/018/1999-04, Air Command and Staff College, April 1999.

[17] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. Readings in Information Visualization: Using Vision to Think. The Morgan Kaufmann series in interactive technologies. Morgan Kaufmann Publishers, San Francisco, Calif, 1999.

[18] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). 1990.

[19] Yan Chen, David Bindel, and Randy H. Katz. Tomography-based Overlay Network Monitoring. In Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, pages 216–231, New York, NY, USA, 2003. ACM.

[20] Alain Cohen. Simulating Virtual Circuits in Mobile Packet Radio Networks. PhD thesis, Massachusetts Institute of Technology, 1986.

[21] Anthony R. Collier. Automated Network Mapping and Topology Verification. PhD thesis, Naval Postgraduate School Naval Postgraduate School United States, Naval Postgraduate School Naval Postgraduate School United States, June 2016.

[22] Jacob H. Cox, Joaquin Chung, Sean Donovan, Jared Ivey, Russell J. Clark, George Riley, and Henry L. Owen. Advancing Software-Defined Networks: A Survey. IEEE Access, PP(99):1–1, 2017.

[23] Steven A. Davidson, Mu-Cheng Wang, Sam Mohan, Frank Bronzo, John Zinky, and Jerry Burchfiel. GIG End-to-End Policy Based Network Management: A new approach to large-scale distributed automation. pages 2011–2018. IEEE, November 2011.

[24] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1(1):269–271, December 1959.

[25] Heidi L. Gebhardt. Communication Modulation Simulators: An Assessment. June 1997.

[26] John Harris, Jeffry L. Hirst, and Michael Mossinghoff. Combinatorics and Graph Theory. Undergraduate Texts in Mathematics. Springer New York, New York, NY, 2008.

[27] Manar Jammal, Taranpreet Singh, Abdallah Shami, Rasool Asal, and Yiming Li. Software-Defined Networking: State of the Art and Research Challenges. arXiv:1406.0124 [cs], May 2014.

[28] Martin Junghans. Visualization of Hyperedges in Fixed Graph Layouts. PhD thesis, Brandenburg University of Technology Cottbus, October 2008.

[29] G. Kessler and S. Shepard. A Primer On Internet and TCP/IP Tools and Utilities. 1997.

[30] Mohit V. Lad, Ricardo V. Oliveira, Michael Meisel, and Ryan Braud. Deep path analysis of application delivery over a network, September 2016. International Classification H04L12/24, G06F11/07, H04L12/26, G06F11/00; Cooperative Classification H04L43/0852, H04L43/08, H04L43/062, H04L43/045, H04L41/22, H04L41/042, H04L41/0631, G06F11/079.

[31] Earl Lawrence, George Michailidis, Vijayan N. Nair, and Bowei Xi. Network tomography: A review and recent developments. In In Fan and Koul, Editors, Frontiers in Statistics, pages 345–364. College Press, 2006.

[32] Ed M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). 2010.

[33] Daniel C. McBride. Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) Architecture and Framework. PhD thesis, Naval Postgraduate School, Monterey, CA, June 2015.

[34] Cindy E. Moran. Defense Information Systems Network Forecast to Industry. Technical report, Defense Information Systems Agency, August 2008.

[35] Donald R. Morrison. PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric. J. ACM, 15(4):514–534, October 1968.

[36] J. Moy. OSPF Version 2. 1998.

[37] National Research Council. Network Science. The National Academies Press, December 2005.

[38] Packet Design, Inc. Enhancing Network Monitoring with Route Analytics, 2013.

[39] Ed R. Enns. NETCONF Configuration Protocol. 2006.

[40] Anna Ritz, Brendan Avent, and T. M. Murali. Pathway Analysis with Signaling Hypergraphs. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 14(5):1042–1055, September 2017.

[41] Keijo Ruohonen. Graph Theory, 2013.

[42] Daniel A. Schult. Exploring network structure, dynamics, and function using NetworkX. In In Proceedings of the 7th Python in Science Conference (SciPy, pages 11–15, 2008.

[43] Rob Shakir, Anees Shaikh, and Marcus Hines. Consistent Modeling of Operational State Data in YANG. Internet-Draft draft-openconfig-netmod-opstate-01, Internet Engineering Task Force / Internet Engineering Task Force, July 2015. Work in Progress.

[44] Joel Sommers. Pytricia: A library for fast IP address lookup in Python, October 2017.

[45] Michael Sperberg-McQueen, Jean Paoli, François Yergeau, Eve Maler, and Tim Bray. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3c recommendation, W3C, November 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[46] United States Marine Corps. Marine Corps Doctrinal Publication 1: Warfighting, 1997.

[47] Y. Vardi. Network Tomography: Estimating Source-Destination Traffic Intensities from Link Data. Journal of the American Statistical Association, 91(433):365–377, March 1996.

[48] Douglas West. Introduction to Graph Theory (2nd Edition). Prentice Hall, September 2000.

[49] Lih-Chyau Wuu, Kuo-Ming Chen, and Tzong-Jye Liu. A longest prefix first search tree for IP lookup. In IEEE International Conference on Communications, 2005. ICC 2005. 2005, volume 2, pages 989–993 Vol. 2, May 2005.

[50] Geoffrey G. Xie, Jibin Zhan, David A. Maltz, Hui Zhang, Albert Greenberg, Gisli Hjalmtysson, and Jennifer Rexford. On static reachability analysis of IP networks. In In Proc. IEEE INFOCOM, 2005.

[51] Sergey Yakovlev. Netdev: Asynchronous multi-vendor library for interacting with network devices, October 2017.

[52] Zelealem Belaineh Yilma. Results in Extremal Graph and Hypergraph Theory. PhD thesis, Carnegie Mellon University, 2011.

[53] T. Ylonen and Ed C. Lonvick. The Secure Shell (SSH) Protocol Architecture. 2005.

[54] Hongyi Zeng, Shidong Zhang, Fei Ye, Vimalkumar Jeyakumar, Mickey Ju, Junda Liu, Nick McKeown, and Amin Vahdat. Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pages 87–99, Seattle, WA, 2014. USENIX Association.

Appendix A

Tools & Libraries

The following tools, libraries, and languages were used in this research. The project summary is referenced directly from each project's website.

A.1 D3

D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

Url: https://d3js.org

A.2 ECMAScript

Since publication of the first edition in 1997, ECMAScript has grown to be one of the world's most widely used general-purpose programming languages. It is best known as the language embedded in web browsers but has also been widely adopted for server and embedded applications.

ECMAScript is based on several originating technologies, the most well-known being JavaScript (Netscape) and JScript (Microsoft). The language was invented by Brendan Eich at Netscape and first appeared in that company's Navigator 2.0 browser. It has appeared in all subsequent browsers from Netscape and in all browsers from Microsoft starting with Internet Explorer 3.0.

Url: https://www.ecma-international.org/ecma–262/8.0/index.html

A.3 Netdev

Netdev is an asynchronous multi-vendor library for interacting with network devices that was inspired by netmiko. Netmiko is a multi-vendor library to simplify Paramiko SSH connections to network devices. Paramiko is a native Python SSHv2 protocol library.

Url: http://netdev.readthedocs.io

Url: https://github.com/ktbyers/netmiko

Url: http://paramiko.org

A.4 NetworkX

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. With NetworkX you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more.

Url: https://networkx.github.io

A.5 NTC Ansible

Multi-vendor Ansible Modules for Network Automation. The TextFSM templates from this package were used in this research to parse the CLI response from Cisco routers on the live network.

Url: https://github.com/networktocode/ntc-ansible

Url: https://github.com/networktocode/ntc-templates/tree/master/templates

A.6 Python

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high-level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on Windows 2000 and later.

Url: https://www.python.org

A.7 PyTricia

Pytricia is a new python module to store IP prefixes in a patricia tree. It's based on Dave Plonka's modified patricia tree code, and has three things to recommend it over related modules (including py-radix and SubnetTree):

- it's faster
- it works in Python 3
- there are a few nicer library features for manipulating the structure.

Url: https://github.com/jsommers/pytricia

A.8 TextFSM

A Python module which implements a template based state machine for parsing semi-formatted text. Originally developed to allow programmatic access to information returned from the command line interface (CLI) of networking devices. The engine takes two inputs - a template file, and text input (such as command responses from the CLI of a device) and returns a list of records that contains the data parsed from the text.

Url: https://github.com/google/textfsm