Available online at www.sciencedirect.com

ScienceDirect

Future Generation Computer Systems 00 (2017) 1–10

FGCS

# A Programmable Policy Engine to Facilitate Time-efficient Science DMZ Management

Chen Xu[1], Peilong Li[1], Yan Luo[1]

*University of Massachusetts Lowell, One University Ave, Lowell, MA, USA, 01854*

**Abstract**

The Science DMZ model employs dedicated network infrastructures and advanced software techniques for large-volume scientific research traffic flows targeting high-throughput and low-latency data transfer. However, current Science DMZ framework lacks of efficient means of user-intent expression and suffers from slow service-delivery due to the manual work involved in the management loop. As a result, a programmable interface that facilitates user-administrator communication in a time-efficient manner is highly demanded. In this paper, we introduce FLowell, an enhanced SDN-powered Science DMZ model deployed on our campus network. Moreover, we propose a programmable policy engine atop the SDN controller that allows network administrators to implement configuration policies in order to manage the network, while simultaneously offering rapid response time network resource request policies for end users. Our experiment results show that user intent in FLowell can be responded and serviced within 1 second. In addition, FLowell reduces the network latency for the research network path by 35%, and boost the disk-to-disk throughput by up to the 10 Gbps line rate.

*Keywords:* Science DMZ, Software Defined Networking, Policy Engine, Fast Service Delivery

## 1. Introduction

ESnet has proposed the Science DMZ model [1], a scalable model designed for local campus networks, to satisfy the increasing demand for high performance scientific research, including large data transfers and real-time human to application interaction. With Software Defined Networking (SDN) techniques emerging as a preferred solution for network management, decoupling the control plane from the data plane within the context of a Science DMZ network environment enables efficient rule-based network control and significantly reduces the required time for network configuration as well as troubleshooting [2, 3, 4].

Although network administrators need not log into network devices in order to manually apply network configuration changes, when leveraging SDN-based techniques, there exists a pronounced latency between the time a user sends a request for a network resource and the time the request is serviced by the network. In this case, inefficiencies continue to endure in the interaction between users and network administrators, which indicates an ongoing need for manual intervention. For example, the University of Massachusetts Lowell (UML) campus IT department constitutes a Change Advisory Board (CAB). Network administrators from CAB are mandated to meet every Thursday to approve or deny changes. All requests to CAB must be submitted by Monday for review on Thursday in the same week. Most changes need to be reviewed off hours. Similar situation exists in Massachusetts Green High Performance Computing Center (MGHPCC) [5] as the administrators work on the requests every Tuesday. In order to accelerate this inefficient interaction, the entire process must be automated, which in turn requires the consideration of four key requirements. Firstly, in order to replace more traditional methods, such as phone calls or emails, users will require a more effective method to submit their requests. Secondly, users should have the ability to submit high level requests that do not depend on intricate knowledge of the low level details regarding the design of the network. Specifically, users should only have to specify the source and destination for the corresponding network resource request. As a result, user requests will have to be converted into detailed network configuration settings for the appropriate switches. Thirdly, understanding there should be a rapid response time, a given user's request should be serviced in a manner that reduces the required processing time as well as the amount of manual labor carried out by a network administrator. Lastly, although such a process will automate some portion of

*Email addresses:* Chen_Xu@student.uml.edu (Chen Xu), Peilong_Li@uml.edu (Peilong Li), Yan_Luo@uml.edu (Yan Luo)

a network administrator's work, the process itself still needs to be under the control and management of the administrator.

In this paper, we introduce FLowell, an enhanced network infrastructure consisting of a SDN-based Science DMZ that supports various forms of data-driven scientific research conducted at UML. Furthermore, FLowell is designed to accelerate large data transfers from UML to MGHPCC, which will be demarcated from the general purpose, campus production network. Moreover, we propose a programmable policy engine on top of the control plane to allow science research teams from different departments the ability to access Science DMZ resources under the purview of FLowell. In this case, the overarching goal is to reduce the requirements in terms of the time to service a resource request as well as the level of manual intervention on the part of a network administrator.

In this paper we make the following contributions. 1) We design a set of simple and human-readable policy rules for defining the network data paths between end hosts and network resources. 2) We design a policy engine that provides users with policy rules for rapid response time network resource requests, independent of whether the requests can be serviced. 3) We design a policy engine that allows administrators the ability to manage data paths irrespective of whether they were generated from a user request or from a newly defined network administrator policy. 4) We design a policy engine that converts policy rules to OpenFlow rules so that the low level knowledge of the underlying network infrastructure is transparent to end users.

The remainder of the paper is organized as follows. In Section 2, we introduce the ESnet Science DMZ model along with our FLowell Science DMZ deployment. In Section 3 we provide the underlying motivations for our work. Subsequently, we describe the policy engine in Section 4. Afterwards, we evaluate the performance of our work within Section 5. Furthermore, in Section 6 we provide a survey of work related to FLowell. Finally, we provide our conclusions in Section 7.

## 2. Background

### 2.1. ESnet Science DMZ Model

Networks in research institutions and organizations normally service two types of traffic, specifically operational business related traffic and scientific research related traffic. However, the majority of the existing campus networks are optimized for business operations, which are incapable of providing low latency, real-time transfers for large scale data. The lack of support for such large scale data transfers within the context of today's network infrastructure serves as a prominent obstacle that hinders the realization of numerous scientific research objectives. As a result, ESnet proposed a Science DMZ model [1] in order to overcome the aforementioned challenges. The Science DMZ model accomplishes this by separating the specifically engineered high-performance data-intensive science network, i.e. the Science DMZ, from the general-purpose network. As a result, each portion of the network can be optimized without interfering with the other.

Figure 1 [6] presents the Software-Defined Networking (SDN) based Science DMZ reference architecture proposed by ESnet to facilitate the flexible provisioning and routing of network flows. This reference model relies upon OpenFlow switches to manage and differentiate the various network flows, while simultaneously enforcing network security policies.

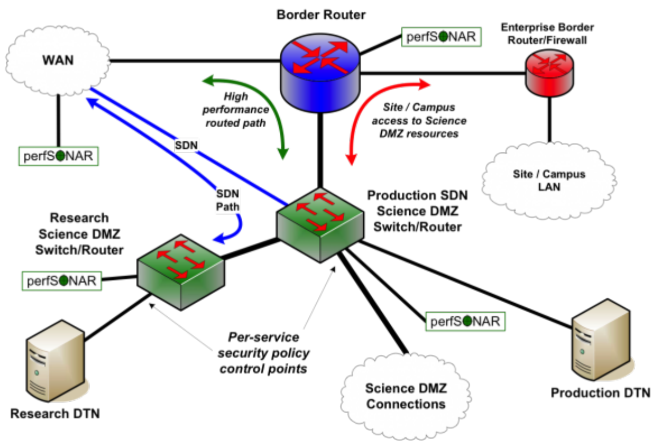

Border Router
perfSONAR
Enterprise Border Router/Firewall
WAN
High performance routed path
Site / Campus access to Science DMZ resources
SDN
perfSONAR
Site / Campus LAN
SDN Path
Production SDN Science DMZ Switch/Router
Research Science DMZ Switch/Router
perfSONAR
perfSONAR
Per-service security policy control points
Science DMZ Connections
Production DTN
Research DTN

Figure 1: ESnet Science DMZ Reference Architecture

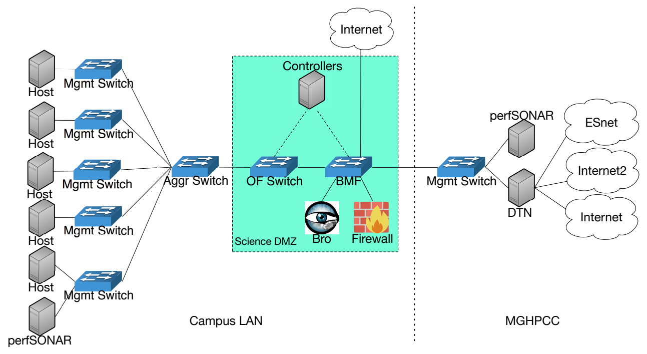


Figure 2: FLowell Science DMZ Network

### 2.2. FLowell Science DMZ

FLowell refers to ESnet Science DMZ model and leverages existing campus cyberinfrastructure resources such as the campus data center, computing clusters located at research laboratories, along with a programmable network test bed consisting of network processors to realize our Science DMZ on campus. As illustrated in Figure 2, the 10 Gbps network connection from UML to MGHPCC serves to bridge on campus researchers with a massive pool of shared computing resources, while the 10 Gbps Layer 2 connection provides a gateway to national research and education networks including Internet2 and ESnet.

#### 2.2.1. UML Campus Network

The campus network is designed to meet the following objectives. 1) A host that is assigned a private IP address, but not a public IP address, should be allowed to access both public and private networks. 2) The network should be able to dynamically filter out and redirect large data flows from the full set of traffic.

3) The network should provide a full 10 Gbps optic-fiber path for all data transfers. Cognizant of the previously stated goals, we deploy the following elements into our campus network.

**Management Switch** – Management switches are currently installed in on campus buildings, and directly face the end hosts in the research laboratories. In addition, the management switches serve to transfer research data from UML to MGHPCC, where the information is stored within a Date Transfer Node (DTN). Furthermore, the management switches provide the opportunity for further scalability in the event that the number of hosts should increase in the future. The high availability (HA) enabled management switches also ensures zero downtime of the network operation.

**Aggregate Switch** – An aggregate switch is included in our design in order to 1) build the connectivity between buildings to allow the data transfers in campus Local Area Network (LAN) without need for SDN control; 2) accumulate traffic from on campus buildings and send the resulting traffic to the OpenFlow switch through a single output port. The aggregate switch serves to overcome the lack of support for the `FLOOD` action by the OpenFlow switch. Given that all traffic emanates from a single port, rather than multiple ports, the aggregate switch serves to minimize the number of wildcard rules and conflicts in a coarse-grained manner, particularly with reference to the number of rules for handling ARP packets.

**OpenFlow Switch** – An OpenFlow switch forwards packets to different destinations based upon the appropriate flow rules. We define the flows into two types, namely an Elephant Flow if it is large continuous data flow and the destination is MGHPCC, and a Mice Flow if the flow needs to go to the Internet.

**Big Monitoring Fabric** – We deploy a Big Monitoring Fabric (BMF) network packet broker to operate within inline mode [7] with service chains designed to enable network admins to easily deploy and manage inline security or analytics tools so as to ensure the resilience against network or tool failures. We create two chains, namely (an *Elephant Chain* and a *Mice Chain*) for the two types of flows mentioned above. For the *Mice Chain* we apply a series of services including a Bro, a firewall, as well as a NAT (introduced in the next paragraph).

**Software** – The following software solutions are leveraged for the purposes of network control, network monitoring, network security and network measurement. The OpenDayLight (ODL) controller [8] and the Big Mon Controller [7] manage the control plane for the OpenFlow switch and the BMF respectively. The Bro [9] IDS is a powerful network analysis framework. In collaboration with the ODL controller, the network security monitor can perform packet analysis, determine the flow type of the packet, and forward the packet to the appropriate destination by calling the correct ODL API in order to install the corresponding rule in the OpenFlow switch. Furthermore, pfSense [10] serves as an open source firewall which can secure the private network while providing NAT services for private IP addresses. Moreover, perfSONAR [11], widely used in the context of science networks, provides our design with the necessary network performance measurement infrastructure.

On the start of the system, the ODL controller connects to the appropriate OpenFlow switches. Then, the Bro IDS begins to monitor the network activities of the *Mice Chain*, while managing the wildcard rules for ARP packet handling and ensuring that packets forwarded to the *Mice Chain* are installed in the appropriate OpenFlow switch.

As shown in Figure 3, once a host starts sending packets: ① The Mgmt. & Aggr. switches forward the traffic to OpenFlow switch. ② As a result of the pre-installed rules the traffic will be forwarded to the *Mice Chain*. ③ The BMF fabric spans the traffic to the Bro IDS, which captures the packet's source IP along with the destination IP and decides if the flow is an Elephant Flow based on a white list table lookup. ④ If the destination IP address corresponds to a Mice Flow destined for the Internet, the Bro IDS will perform no action and the traffic will pass through firewall. The firewall will NAT the private source IP address to a public IP address. ⑤ The traffic is sent to the public network. ⑥ If the destination IP address is part of the MGHPCC network, the Bro IDS will call the appropriate ODL API. ⑦ The ODL controller will install the corresponding flow rules into the OpenFlow switch. ⑧ Subsequent packets will be re-routed to the *Elephant Chain*. ⑨ Packets part of the *Elephant Chain* will be forwarded to MGHPCC.

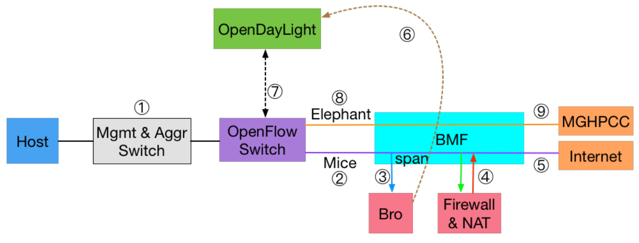


Figure 3: FLowell Network Flow

### 2.2.2. *MGHPCC Network*

The MGHPCC network serves two functions, namely 1) hosting a Data Transfer Node (DTN) in order to store large scientific, research data and 2) sharing the research data with other sites including, for example, AL2S [12] and Internet2 [13]. In MGHPCC, we deploy a Dell PowerEdge R730 server with storage directly attached to it as our DTN. The DTN has a private 10Gbps connection for data transfers to or from campus and a public 10Gbps connection for sharing data with the Internet and other sites. Two different data transfer tools are served on these networks. Globus serves to manage data transfers over the public network, whereby a Globus Connect Server (GCS)[14] configures the DTN as a Globus endpoint. Globus enables all users with local accounts on the DTN to share and transfer files to or from this endpoint. Users need to install the Globus Connect Personal (GCP)[15] client onto their computer in order to connect to the GCS server. Fast Data Transfer (FDT)[16] serves to manage data transfers over the private network. FDT has the capability to perform efficient data transfers at disk speed over networks with standard TCP. Globus and FDT were selected for their ease of use and for their support of all major platforms.

Table 1: Software Deployed in FLowell Science DMZ

| Software | Function | Description |
|---|---|---|
| OpenDayLight Controller | Network Control | The control plane for OpenFlow switch |
| Big Mon Controller | Network Control | The control plane for BMF |
| Bro | Network Monitor & Analysis | Monitor the network traffic and filter Elephant Flow from it |
| pfSense | Network Security | Protect the LAN from external attack and do NATing for private IP address |
| pfSONAR | Network Measurement | Provide some active network measurement tools |

## 3. Motivation

Our goal is to build a policy engine, a human-friendly and automated system to dynamically dealing with users' specified network resource request policies. To make this realized, we need to solve the following questions.

### 3.1. *Question 1: How Can We Speedup the Service Delivery Process for an End User's Request?*

Firewalls installed on campus and enterprise networks help prevent malicious external network traffic from entering LANs. In the context of an SDN-powered Science DMZ, an OpenFlow switch functions as a Layer 2 firewall as packets need to match rules installed within the switch in order to pass through the switch. The pre-installed firewall and OpenFlow rules can only cover the most basic of network activities, such as website navigation as well as sending or receiving emails. However, if a network user wants to access a particular external resource that is restricted by the current rule set, then the user must request access from a network administrator. The user-admin interaction process is normally carried out by way of email or phone calls, which is not only extremely inefficient, with resolution typically occurring within the span of hours or days, it lacks strong accountability.

To remove the human interaction element from the process and speedup the service delivery time, we propose a policy engine design that enables network users and network administrators to work in a time-efficient manner. In particular, end users can submit their network resource access requests via a web graphic interface built on top of our proposed network policy rules. The user can receive an immediate response to their request, either accepted or rejected, depending on whether the request complies with a predefined set of criteria, i.e. a white list, setup by the network administrator.

### 3.2. *Question 2: Why Not Use Static OpenFlow Rules?*

The first question leads us to the next, namely why a white list is employed to respond to a given user's request rather than simply placing static rules into an OpenFlow switch. We answer this newly proposed question from two different aspects depending on the granularity of the rule.

Firstly, if entries in a white list are expressed in a coarse-grained manner in order to match the majority of the user requests, then routing conflicts may occur. As an example, suppose an entry from the white list permits all packets from IP range $R_{IP}$ to pass through the switch. When a user requests a path from $host1$ in $R_{IP1}$ to destination $dst1$ in $R_{IP2}$, while another user requests a path from $host2$ in $R_{IP1}$ to $dst2$ in $R_{IP2}$, both are legitimate requests. However if $dst1$ and $dst2$ are on two different physical ports of the OpenFlow switch, there are no rules to direct the two user requests to the correct output port.

Secondly, an OpenFlow switch is constrained by memory resource limitations that should not be squandered. If the pre-installed static rules are too fine-grained to direct every flow of the network, we may unnecessarily consume considerable memory resources on the switch. Furthermore, copious entries within the switch will increase the flow table lookup time dramatically and adversely effect performance within the latency-sensitive Science DMZ network.

Therefore, a white list is necessary in our solution and the flow rules will be installed in the switch after users submit their requests. What's more, we design a module in our policy engine to check flow conflict dynamically before a flow rule is installed. Thus, only necessary rules can be pushed to OpenFlow switch so as to save the memory space and the rules are fine-grained without conflict.

### 3.3. *Question 3: How Can We Map an End User's Request to a Set of Network Rules?*

In a Science DMZ, an end user typically has no prior knowledge regarding network operation and network hardware configurations. Therefore, from an end user's perspective, it is ideal to simply submit a network resource request and have the system determine the necessary optimal path. For example, an end user first provides the source, host name, and destination, service name, corresponding to the task at hand. Subsequently, the system verifies the legitimacy of the request. If the request is valid the system will find a load-balanced path between the requested source and destination, and provide the number of input and output ports along the path.

To this end, we provide a set of policy rules to help express the user's intention, and design a policy manager inside a policy engine in order to parse the intentions from users as well as to generate the final set of OpenFlow rules. The policy manager interacts with the SDN controller in order to maintain the existing rule set as well as to install any missing rules. Given the capabilities afforded to us by SDN, the centralized controller maintains a global view of the entire network topology. Hence, we can readily find an optimal path in order to satisfy a given user's request.

## 4. Design

As shown in Figure 4, our FLowell Science DMZ infrastructure leverages a policy engine on top of the control plane, which is capable of receiving and imposing policy rules. By leveraging the policy rules and the corresponding policy engine, users can easily request network resources and network administrators can simplify network path configurations by avoiding some forms of manual work. The policy engine consists of a web based GUI, a policy manager and a policy repository. We will discuss the details for each portion of the design in the following subsections.

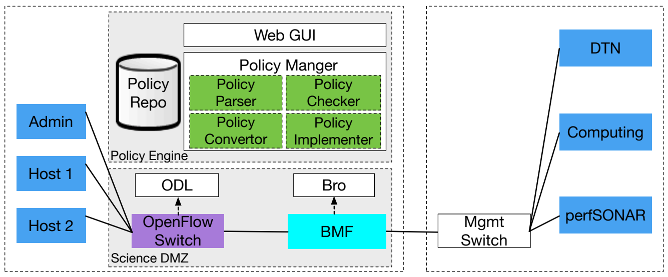


Figure 4: Components of the Policy Engine

### 4.1. Policy Rule

Given that users and administrators may not necessarily have in depth programming knowledge, we design a simple and intuitive policy rule set. The policy rule consists of three keywords, 1) the source IP address, specifying the end host, 2) the destination IP address, representing the network resource, and 3) the flow operation to perform, in terms of whether to establish a new data path or remove an existing one. The resulting keyword combination specifies a request suitable for configuring a particular network path. Table 2 provides details regarding each of the policy rule fields. For the sample policy provided below, a user intends to request that a path be established from 10.0.0.1 to 10.0.0.2.

**Src_IP** : 10.0.0.1,
**Dst_IP** : 10.0.0.2,
**Flow_OP** : *install*

Table 2: Policy Rule Keywords

| Keyword | Value |
|---|---|
| Src_IP | IP address of the end host |
| Dst_IP | IP address of network resource |
| Flow_OP | install, remove |

### 4.2. Policy Manager

In the policy engine, we implement a policy manager as the core component responsible for receiving policies sent by users and network administrators from the web based GUI, parsing the policies, checking for conflicts, determining the shortest forwarding paths, generating the necessary OpenFlow rules and storing the rules to the policy repository. As shown in Table 3, in order to service the various tasks, the policy manager can be divided into four components, namely the policy parser, the policy checker, the policy converter and the policy implementer. We explain the functionality of each block in details as follows, and the pseudo-code demonstration is shown in Algorithm 1.

When the policy manager receives a policy rule from the GUI, it first calls the *policy_parser* function to parse the policy using regular expressions (RegEx), extracts the values of the keywords referenced in Table 2 and check if all values are valid, for example an IP address should be described with the correct format and range. If the policy originates from a user, the policy manager performs a white list look up to verify whether the request is permitted. Afterwards the policy manager verifies whether or not the requested network path is duplicated in the *Flow* array by calling the *policy_checker* function. In this case, we must perform duplicate detection as repeated requests may result in flow rule conflicts within the OpenFlow switch. If the policy originates from a network administrator, the installation operation will skip performing the white list lookup, but will continue to check for duplicates. Furthermore, we validate the existence of a path in the event an administrator would like to remove a path due to some reasons, e.g., malicious activities detected on this path. Subsequently, the policy manager calls the *policy_converter* function to extract the network topology from the ODL controller, constructs a graph of the topology, determines the shortest network path by employing Dijkstra's algorithm and generates a flow object that contains the rules to be installed within the network. The example below illustrates the format of a flow object. For this object, the *path* key contains the user's requested path, while the *connector* key lists the switches as well as the ingresses and egresses along the network path. Subsequently, the *policy_implementer* method is called to update the *Flow* array or the *Pending* array depending on the results of the *policy_checker* function. Figure 5 illustrates the work flow carried out by the policy manager.

$$\textbf{flow\_obj} = \{'path' : ['10.0.0.1', '10.0.0.2'],\\ 'connector' : [['switch1', '1', '2'], [], []...]\}$$

### 4.3. Web-based GUI

We provide a simple and intuitive web based GUI to allow authorized network administrators and end users the ability to easily create and implement policy rules. Within the GUI, we provide a set of forms so that users can fill in the source and destination IP addresses along with the operation to be performed. Users can conveniently issue their policy request via a *submit* button. Furthermore, end users can verify the result of the submitted network resource request through the same GUI interface, while network administrators can view all approved network paths and pending paths in order to better understand, control and manage the network.

### 4.4. Policy Repository

We implement a policy repository to store the white list as well as the flow objects in the *Flow* array along with the *Pending* array. The *Flow* array consists of the permitted paths as

Table 3: Functions in Policy Manager

| No. | Function | Explanation |
|---|---|---|
| 1 | policy_parser(str policy) | Parse the policy and check for validation |
| 2 | policy_checker (str src_ip, str dst_ip, str flow_op) | Check for feasibility and rule existence |
| 3 | policy_converter (str src_ip, str dst_ip) | Map policy to flow rule(s) |
| 4 | policy_implementer(dict flow_obj,str flow_op) | Install flow rule(s) into the system |

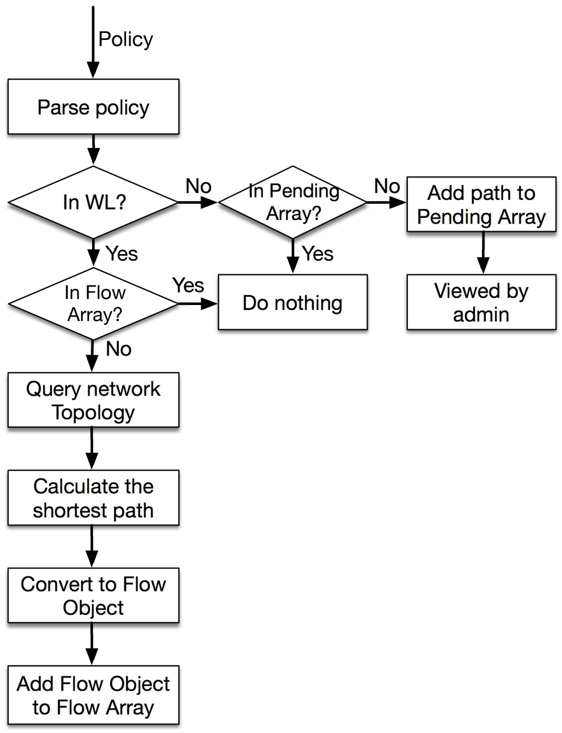


Figure 5: Policy Manager Work Flow

**Algorithm 1** Policy Manager

```
1:  $WL[]$ : array for white list
2:  $Flow[]$ : array for approved path
3:  $Pending[]$: array for pending path
4:  dict $flow\_obj$: flow object
5:
6:  function POLICY_PARSER($policy$)
7:      Use regex to extract values from policy according to keywords
8:      if $all\ values\ are\ valid$ then
9:          return $src\_ip$, $dst\_ip$, $flow\_op$
10:
11: function POLICY_CHECKER($src\_ip$, $dst\_ip$, $flow\_op$)
12:     $flow$ = [$src\_ip$,$dst\_ip$]
13:     if $User\ and\ flow\ in\ WL$ or $Admin$ then
14:         if $flow\_op\ is$ "$install$" $and\ flow\ not\ in\ Flow$ then
15:             $flow\_obj$ = policy_convertor($src\_ip$, $dst\_ip$)
16:             policy_implementer($flow\_obj$, $flow\_op$)
17:         else if $flow\_op\ is$ "$remove$" $and\ flow\ in\ Flow$ then
18:             $flow\_obj$['path'] = $flow$
19:             policy_implementer($flow\_obj$, $flow\_op$)
20:     else if $User\ and\ flow\ not\ in\ WL$ then
21:         $flow\_obj$['path'] = $flow$
22:         policy_implementer($flow\_obj$, $flow\_op$)
23:     return $flag$
24:
25: function POLICY_CONVERTER($src\_ip$, $dst\_ip$)
26:     Call ODL API to get the detail of network topology
27:     Calculate the shortest path for given $src\_ip$ and $dst\_ip$
28:     Determine the in and out ports of each switch in the path
29:     Generate $flow\_obj$
30:     return $flow\_obj$
31:
32: function POLICY_IMPLEMENTER($flow\_obj$, $flow\_op$)
33:     if '$connector$' $in\ flow\_obj\ and\ flow\_op\ is$ "$install$" then
34:         Add $flow\_obj$ to $Flow$
35:     else if flow_op is "remove" then
36:         Remove $flow\_obj$ from $Flow$
37:         Remove corresponding flow(s) from OF switch
38:     else
39:         Add $flow\_obj$ to $Pending$
```

well as flow installation information, while *Pending* array gives the paths that are not listed in the white list a second chance and these paths will be viewed by administrators for further decision. Thus, we increase the error-tolerant rate in case of the request path is reasonable but not in the white list. There are three reasons why the arrays needed to be stored. 1) Upon receipt of a packet the Bro IDS will need to inspect the *Flow* array in order to determine if the packet matches one of the approved paths and if so fetch the corresponding flow installation information. 2) A user will need to check the GUI interface in order to determine if the requested flow has been approved. 3) Administrators will need to inspect the GUI interface in order to collect a comprehensive list of the approved network paths. Policies will be stored in the repository in the following format.

$$\textbf{Flow} = [\{'path' : ['10.0.0.1', '10.0.0.2'], \\ 'connector' : [['switch1', '1', '2'], [], [] ...]\}, \{\}, \{\}...]$$

## 4.5. Policy Engine Control Flow

In this subsection, we discuss how the components of the policy engine collaborate with one other in order to carry out automated network resource control as well as management.

The policy engine control flow, shown in Figure 6, can be divided into the control path for a user and the control path for a network administrator. The difference between the two control flows is that network administrators have the necessary privileges to remove an approved path from the network, while end users can only request that a flow be instantiated.

**End User's Control Path**: An end user, e.g., a researcher, will first create a policy through the web based GUI and send it to the policy manager. The policy manager then parses the policy to determine the next course of action. Upon receipt of a new packet, the Bro IDS verifies whether the source and destination IP address pair is in the *Flow* array within the policy repository. If so, the Bro IDS will fetch the network connection

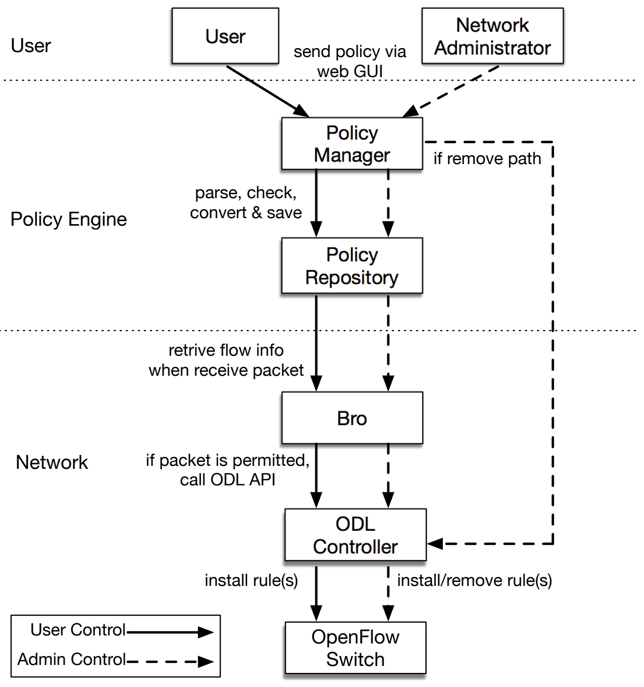


Figure 6: Policy Engine Control Flow

information for the corresponding path and call the ODL API to install the necessary rules into switch(es).

**Administrator's Control Path**: The control path for a network administrator, e.g., a campus network IT or network manager, is similar in nature to an end user's control path, however, if an administrator plans to add a new path or approve a pending one, the policy manager does not perform a white list lookup and directly saves the flow object to the *Flow* array. If an administrator would like to remove a malicious flow which was previously approved, after receiving the policy, the policy manager will first delete the flow object from the *Flow* array to make sure Bro IDS will not keep installing rules in switches and then communicate with the ODL controller in order to remove the corresponding flow.

## 5. Evaluation

In this section, we evaluate our work with two experiments: 1) we evaluate the network performance to illustrate the advantages of the Science DMZ infrastructure; and 2) we evaluate the performance and overhead of the policy engine. As shown in Figure 7, in the first evaluation we set up a DTN node within the local UML campus network and run both latency and throughput experiments with the remote DTN node located within the MGHPCC network. The remote DTN server retains both public and private IP address for external and internal access. We compare the performance metrics with or without employing Science DMZ dedicated path over the network link. In the second evaluation, we implement our policy engine within the server hosting the SDN controllers and measure the policy manager's response time, as shown in Figure 8. The hardware specifications of the machines we used in the two experiments are listed in Table 4.

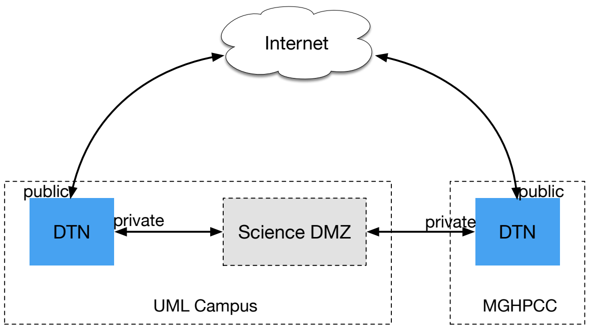


Figure 7: Science DMZ Performance Test Bed

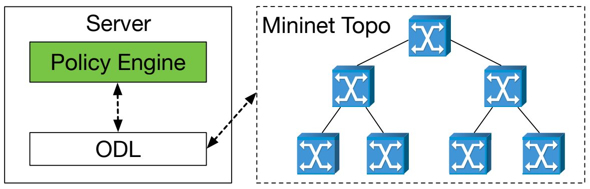


Figure 8: Policy Engine Performance Test Bed

### 5.1. *Latency and Throughput*

Typically, latency refers to the amount of time it takes for data to travel from a sender to a receiver. However, in the case of our experiment we consider the Round-Trip-Time (RTT) as the network latency. The RTT can be simply measured by using the ICMP protocol. For packets traversing the Internet, the average RTT is approximately 5.424 ms. For packets traversing the Science DMZ, the average RTT decreases to approximately 3.483 ms with a decrease rate by 35%. Please note that since both UML campus and MGHPCC locate geographically in Massachusetts, the RTT of the Internet path is not significantly large. However, with the expansion of our network in the future, the benefit of applying Science DMZ will be more salient.

Throughput measures the data transfer rate and serves as an indication as to how much data can be transferred from a given sender to a given receiver within a specified unit of time. The higher the throughput, the lower the transmission delay. In order to measure throughput, we leverage iperf as well as FDT to transfer 10 GB of data between two DTNs with & without the Science DMZ and observe both memory-to-memory and disk-to-disk throughput. In addition, we vary the number of parallel connections and observe the resulting effects on the throughput. Table 5 displays the recorded results. By observation, we know that the throughput of the campus public network is limited to 100 Mbps. By contrast, in the case of the Science DMZ 10 Gbps network, we can achieve almost 100× the memory-to-memory and disk-to-disk throughput when transferring the data

Table 4: Hardware Specification of Machines in the Test Beds

| | DTN | Server |
|---|---|---|
| Model | Dell PowerEdge R730 Server | Dell PowerEdge R730 Server |
| CPU | 2× Intel Xeon E5-2643 3.4GHz 6 Cores | 2× Intel Xeon E5-2643 3.4GHz 6 Cores |
| Memory | 6× 16G RDIMM | 4× 8G RDIMM |
| Hard Disk | 16× 1.8TB SAS Hard Drive | 1× 1TB SATA Hard Drive |
| NIC | Broadcom Corporation NetXtreme II BCM57810 10Gb<br>Mellanox Connect X3 Dual Port 10Gb SFP+ | Broadcom Corporation NetXtreme BCM5720 1Gb |

using 4 parallel connections. The factor "4" in parallelism is a practical number suggested by ESnet [17] to render the optimal line rate performance. In our experiments, when increasing the parallelism factor from 4 to 8, we can only observe a slight gain by 0.1 Gbps in throughput.

Table 5: Throughput Test Results

| Tool | w/ DMZ | File Size | Parallelism | Avg. Throughput |
|---|---|---|---|---|
| iperf | No | N/A | 4 | 95.6 Mbps |
| iperf | Yes | N/A | 4 | 9.40 Gbps |
| FDT | No | 10 GB | 1 | 91.605 Mbps |
| FDT | Yes | 10 GB | 1 | 6.681 Gbps |
| FDT | No | 10 GB | 4 | 91.792 Mbps |
| FDT | Yes | 10 GB | 4 | 9.191 Gbps |

### 5.2. The Performance and Overhead of the Policy Manager

In Algorithm 1, three factors will affect the response time once the policy manager receives a new rule. Firstly, the size of the white list determines how long the policy manager will take to approve the path request. Secondly, the size of the *Flow* array affects the processing time to check for path conflicts. Thirdly, the number of switches and the network topology determines the time to build the graph and find the appropriate flow path.

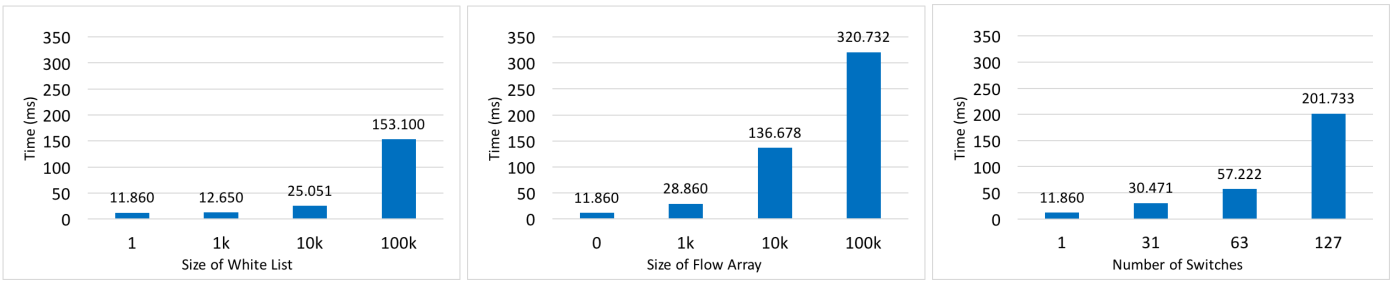
To study how each factor affects the response time, we first define the baseline experiment scenario. As the baseline, we initialize the white list with a single path in order to ensure that all requests will be approved. We also create an empty *Flow* array so that duplicated checks will be passed, and we only instantiate a single OpenFlow switch connected to two end hosts in order to reduce the time to build a graph and find the shortest path. With the baseline, we study the performance effect of three different factors in the policy manager by changing only one factor at a time. Figure 9 demonstrated the collected results. In the case of the base line the observed response time is 11.86 ms. In Figure 9 (a), we increase the number of paths in the white list from 1 to 1k, 10k and 100k. In order to maximize the search time, we organize the list such that the matching path is always the last entry. For a white list size of 100k entries, the response time is 153.1 ms. In the case of Figure 9 (b), we focus on varying the number of objects in the *Flow* array. Each new object can simply be appended to the array, given that there should be no duplicates. For an array size of 100k elements the observed response time is 320.732 ms. In Figure 9 (c), we vary the complexity of the network topology. We use Mininet[18] to simulate the network environment and apply fat-tree topologies consisting of between 5-7 layers. The response time for a 7 layer fat-tree topology, consisting of 127 switches, is approximately 201.733 ms. In addition, we test the performance of the policy manager in the case where the white list and the *Flow* array consist of 100k entries each and the topology consists of a 7 layer deep fat-tree topology. The average response time in this scenario is 619.561 ms. For each of the above four experiments, although the response time increased to hundreds of milliseconds, it still can be considered to be of significantly short duration. We conclude that our policy manager is both scalable and can immediately respond to a user's request.

## 6. Related Work

Network management continues to be an ongoing challenge. SDN attempts to simplify this challenge by offering a clear separation between the control plane and the data plane. In recent years a significant amount of work has been done in applying SDN related technologies towards network management.

Balas et al. [4] proposed a system named SciPass, which is implemented with OpenFlow and the Bro IDS system in order to increase the security and performance of their Science DMZ environment. In the SciPass system, the authors applied a load balancer to divide the traffic into a set of IDS sensors. Furthermore, the system can help filter flows, such as large data-intensive flows, which can then be configured to bypass the firewall in order to improve performance. Although FLowell similarly filters out Elephant flows from the comprehensive set of network traffic, the system includes a policy engine and a set of policy rules in order to provide users and network administrators with the ability to request, configure, and approve such large data flows.

Kim et al. [2] proposed to implement a policy layer on top of an SDN controller in order to allow network operators the ability to create network policies in the Procera [19] language. By leveraging policies, their work was shown to reduce the workload of network configuration and management within the context of campus and in-home networks in response to changes in the current network state, such as, for example, increases in the data transfer rate. Although their proposal leverages flow fields within their policy language in order to facilitate changes in network behavior, our work focuses upon accelerating the service delivery process for end users with the intervention of admins. More importantly, our policy rules are more intuitive and assume neither technical knowledge nor the



(a) (b) (c)

Figure 9: Policy Manager Response Time

understanding of network configuration from the users/admins comparing with the Procera language.

PolicyCop [20] was proposed by Bari et al. as an open and flexible QoS policy management framework for SDN. The framework allows specifying QoS-based Service Level Agreements (SLAs) and enforcing them by leveraging the SDN controller's API. By contrast, FLowell focuses upon simplifying the network resource request and response process while simultaneously assisting network administrators with managing the accessibility of resources by end users.

Lara et al. [3] proposed an OpenFlow-based framework that allows the network operator to create and implement network security policies. For the given framework and the corresponding set of policies, flows can be spanned and sent to different services, such as an IDS as well as a spam detection system. Once the security services alert upon a set of malicious activities, the corresponding policies specify the necessary actions to perform. While the focus of the paper is related with network security threat detection, the primary goal of FLowell is upon accelerating network resource management in response to end user requests, while simultaneously offering end users the ability reserve paths within the network and indirectly change the flow state within the switches.

## 7. Conclusion

In this paper, we first introduce our FLowell Science DMZ infrastructure, which is designed to overcome the shortcomings in our current campus network in order to accelerate large-volume data transfers. With the dedicated Science DMZ path, we are able to achieve reducing latency from 5.4 ms to 3.5 ms while simultaneously increasing throughput from 91.8 Mbps to 9.2 Gbps. As a result, the file transfer time is reduced significantly, particularly for large file sizes. More importantly, we propose a policy engine on top of the network control plane. The policy engine enables network users to submit demands for network resources, which, for valid requests, are automatically and immediately serviced. As a result, user to administrator interactions, specifically in terms of the network resource request approval process, are simplified and can be finished in 1 second. At the same time, network administrators can leverage policy rules to manage the data paths within the network.

## Acknowledgment

We acknowledge the technical assistance from University of Massachusetts Lowell IT department and the Massachusetts Green High Performance Computing Center (MGHPCC). This work is supported in part by the National Science Foundation (No. 1547428, No. 1738965, No. 1440737 and No. 1450996) and a grant from Intel Corporation.

## References

[1] ESnet Science DMZ.
URL `https://fasterdata.es.net/science-dmz`

[2] H. Kim, N. Feamster, Improving network management with software defined networking, IEEE Communications Magazine 51 (2) (2013) 114–119.

[3] A. Lara, B. Ramamurthy, Opensec: Policy-based security using software-defined networking, IEEE Transactions on Network and Service Management 13 (1) (2016) 30–42.

[4] E. Balas, A. Ragusa, Scipass: a 100gbps capable secure science dmz using openflow and bro, in: Supercomputing 2014 conference (SC14), 2014.

[5] Massachusetts Green High Performance Computing Center.
URL `https://www.mghpcc.org/`

[6] ESnet Science DMZ Software Defined Networking Reference Architecture.
URL `https://fasterdata.es.net/science-dmz/software-defined-networking/`

[7] Big Monitoring Fabric Inline.
URL `http://go.bigswitch.com/rs/974-WXR-561/images/BMF\%20Inline\%20-\%20DMZ\%20Deployment\%20White\%20Paper.pdf`

[8] OpenDayLight.
URL `https://www.opendaylight.org/`

[9] Bro.
URL `https://www.bro.org/`

[10] pfSense.
URL `https://www.pfsense.org/`

[11] perfSONAR.
URL `https://www.perfsonar.net/`

[12] AL2S.
URL `https://www.internet2.edu/products-services/advanced-networking/layer-2-services/`

[13] Internet2.
URL `https://www.internet2.edu/`

[14] Globus Connect Sever (GCS).
URL `https://www.globus.org/globus-connect-server`

[15] Globus Connect Personal (GCP).
URL `https://www.globus.org/globus-connect-personal`

[16] Fast Data Transfer (FDT).
URL `http://monalisa.cern.ch/FDT/`

[17] Fast Data Transfer Tool (FDT).
URL `https://fasterdata.es.net/data-transfer-tools/fast-data-transfer-tool-fdt/`

[18] Mininet.
URL `http://mininet.org/`

[19] A. Voellmy, H. Kim, N. Feamster, Procera: a language for high-level reactive network control, in: Proceedings of the first workshop on Hot topics in software defined networks, ACM, 2012, pp. 43–48.

[20] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, Policycop: An autonomic qos policy enforcement framework for software defined networks, in: Future Networks and Services (SDN4FNS), 2013 IEEE SDN For, IEEE, 2013, pp. 1–7.