Data-Centric Publish/Subscribe Routing Middleware for Realizing Proactive Overlay Software-Defined Networking

Akram Hakiri
CNRS; LAAS, 7, avenue du Colonel Roche
Univ de Carthage, SYSCOM ENIT, ISSAT
Mateur, Tunisia
Hakiri@laas.fr

Aniruddha Gokhale
Institute for Software Integrated Systems, Dept of EECS
Vanderbilt University, Nashville, TN 37203
a.gokhale@vanderbilt.edu

ABSTRACT

Software Defined Networking (SDN) has emerged as an attractive solution to allow cloud-to-cloud interconnection and federation. SDN technologies, such as OpenFlow, use both reactive hop-by-hop and proactive approaches to program the switches. The reactive strategy incurs substantial scalability problems for large networks due to the hop-by-hop behavior while the proactive approach is hard to implement in practice due to the need to forecast all possible forwarding rules ahead-of-time. An attractive and more realistic alternative is the proactive overlay SDN approach, however, many challenges must first be overcome to realize it. Existing techniques to program the switches use low-level programming abstractions, which are error-prone and cannot scale. Middleware-based solutions, e.g., using XMPP, are stateful and hence also incur substantial scalability issues. Although content-based publish/subscribe (pub/sub) solutions have been used in the past for SDN, they rely on brokers, which is problematic and incurs unnecessary additional infrastructure elements that pollute the SDN architecture. To address these issues, this paper demonstrates how the strengths of the data-centric, broker-less pub/sub paradigm can be exploited to realize proactive overlay SDN for inter cloud domain federation. To that end, we first present the design rationale and architecture of our solution called POSEIDON (Proactive brOkerless SubscribEr Interest-Defined Overlay Networking). Second, we present the messaging protocol between the controller and switches. Finally, we present results of evaluating POSEIDON and illustrate how it improves data delivery and provides high performance at the network-level in proactive overlay SDN.

CCS Concepts

•**Networks** → *Routing protocols; Overlay and other logical network structures;* •**Software and its engineering** → *Message oriented middleware;*

DEBS '16, June 20-24, 2016, Irvine, CA, USA

DOI: http://dx.doi.org/10.1145/2933267.2933314

Keywords

Publish/Subscribe, Overlay Network, Software-defined Networking, Network Virtualization.

1. INTRODUCTION

Service providers are increasingly using data centers for offering services such as video steaming. This approach has been validated by the success of services, such as YouTube and Netflix. This trend is further growing with the advent of Internet of Things (IoT), where the applications are making increasing demands on computation, storage, and network resources, in turn forcing service providers to consolidate groups of data centers in the form of cloud domains. The key requirement is that resources – in our case the network – should be made available any where and any time. Unfortunately, the current network architecture is not suited to support the elastic and on-demand networking resource demands stemming from the federation of cloud domains.

Software Defined Networking (SDN) [23] has emerged as a promising technology for network management, e.g., to provide new capabilities for managing individual flows, automate resource provisioning, and provide more agility in deploying, configuring and instantiating cloud networking. The separation of the packet forwarding logic from the data plane, i.e. from the switches, and handing it to an external controller, which embeds all the network intelligence, makes it easier to manage and control the switches and networks since any change must now be made only in the controller.

OpenFlow [20] is a dominant SDN technology that allows the controller to install flow rules in the underlying switches it controls to enable packet forwarding. OpenFlow provides both reactive and proactive approaches to setup flow rules in the switches. In the reactive flow setup, for every new flow that arrives at a switch, that switch should send a *packet-in* event to the controller so that the latter can send the switch all the flow rules that it should apply to any incoming packets belonging to that flow. This behavior is illustrated by every switch that lies in the flow path. The controller contacts all the switches between the source and destination of a flow in a point-to-point manner to install the desired forwarding rules.

The reactive approach incurs several performance and scalability limitations when the number of switches in a flow increases, which is often the case when connecting distributed cloud domains. The problem is further exacerbated for fine-grained flow matching in latency-sensitive, long-lasting short flows because of an explosion in the number of forwarding states [27]. To address the overhead stemming from this

hop-by-hop behavior of the reactive approach, the proactive flow setup approach was conceived. In this approach, the controller populates all possible flow rules into the switch ahead-of-time for all possible traffic that could arrive at that switch. This behavior is akin to typical routing tables in traditional networks.

Although the proactive flow setup approach tries to provide better performance by avoiding the extra-latency of the first packet matching [4], it is hard to realize in practice because of the need to forecast all possible flow entries that can match against future arriving packets. Moreover, irrespective of the approach used, the OpenFlow switches must buffer a number of short flows in the flow table. However, typical buffer sizes do not exceed a few hundreds of packets thereby causing memory overflows. Subsequently, an extra latency may be incurred in forwarding packets across large networks as the number of new flows to be programmed increases and the SDN controller has to setup all the properties for those flows onto the SDN switches it controls.

With the advent of Internet of Things and cloud federations, it is becoming increasingly important to be able to interconnect multiple cloud domains. Although the proactive SDN approach is promising, as discussed above, the concept is mostly of theoretical value and hard to implement in practice. To meet these urgent needs, we surmise that the easy-to-implement functionality of the reactive approach can be blended into the proactive approach, however, without incurring the scalability issues of the reactive approach by enabling the proactive SDN at an overlay layer. The overlay property stems from the ability to use multiple tunnels to create network slices whose endpoints terminate in a variety of entities, such as virtual switches or physical edge routers. In such an overlay approach, only the endpoint switches of the tunnels must be programmed by the controller instead of every switch in the flow path.

A number of challenges manifest in this proposed approach. The first challenge stems from having to decide which technique to use in the SDN controllers and switches to program the endpoints of the tunnels that reside in the forwarding plane. One approach is to leverage existing OpenFlow APIs. However, these APIs are too low-level and lack intuitive abstractions that can support compositional semantics [10] to combine a sequence of rules and process them in parallel. Middleware solutions have tried to address these issues. For instance, some approaches have adopted the Extensible Messaging and Presence Protocol (XMPP) [19] to distribute the control plane and management plane information to end servers. XMPP, however, is a stateful protocol that requires a server to retain history information status carried over from the previous session processing. Subsequently, XMPP creates significant overhead of data when multiple users are interconnected and thereby suffers from scalability issues.

The authors in [15] introduced line-rate, content-based middleware and matching semantics to disseminate routing information to SDN-enabled switches. Similarly, authors in [1] and [29] introduced content-based middleware to enable group communication as well as control and management tasks among distributed SDN controllers, respectively. However, a significant limitation of content-based routing systems is the lack of decoupling between publishers and subscribers for distributed SDN. In this approach, both participants remain tightly coupled to a broker for message delivery. Moreover, the broker cannot guarantee message delivery to applications, which may require additional enforcement outside of the existing middleware solutions.

To address these concerns, we propose Proactive brOkerless SubscribEr Interest-Defined Overlay Networking (POSEIDON), which is a broker-less, extensible architecture for proactive overlay SDNs that can be realized within OpenFlow. POSEIDON is a publish/subscribe (pub/sub) middleware that exploits the expressiveness power of the data-centric information model to allow the controller to configure and manage the data plane, and to describe the states of the network elements, e.g. creating route instances, exchanging routes and VPN membership information, and managing link status for connection and disconnection. POSEIDON requires proactively programming only the endpoints of the tunnel at boot-time or at policy creation-time. The data plane can thereafter populate packets over these overlay tunnels. The result is that all packets are forwarded at line-rate, which eliminates any unnecessary latency induced by consulting the controller for every flow thereby improving scalability. To overcome the low-level programming issues, POSEIDON uses the Pyretic [25] description language to offer a higher-level of abstraction that is both intuitive and hides the complexity of the underlying network.

The following benefits accrue from our work:

- POSEIDON has full configuration and control of how it processes and forwards packets through the SDN data plane.
- When upgrading the OpenFlow-enabled switches, POSEIDON can continue to perform the forwarding operation independently from the underlying firmware without deteriorating network performance.
- The use of a higher-level of abstraction enables POSEIDON to provide a combination of fine-grained flows at the virtual edge switches and coarse-grained flows in the physical underlay core network devices.
- The use of pub/sub middleware provides higher levels of structure and semantics to build common and computation-independent services.

The remainder of this paper is organized as a follows: Section 2 uses two concrete real-world case studies to elicit the solution requirements for POSEIDON; Section 3 describes the design of POSEIDON and its messaging protocol illustrating how they meet the solution requirements; Section 4 shows how the POSEIDON architecture can address the needs of our motivational case studies; Section 5 presents results of experiments evaluating various properties of POSEIDON; Comparison with related work is described in Section 6; and finally, conclusions drawn from this work and lessons learned are presented in Section 7.

2. MOTIVATIONAL CASE STUDIES

This section describes motivating use cases, which are used to elicit key requirements that must be met by POSEIDON.

2.1 Use Case 1: Data Center Networking and Virtualization

High bandwidth multimedia applications like YouTube and Netflix are typically hosted over distributed data centers

connected over different ISPs which may use data replication on different sites to ensure the availability and reliability of their services. The traffic generated at an ISP's edge network may be aggregated over virtual tunnels with a certain bandwidth and forwarded to end users for performance and efficiency sake. Accordingly, applications should be able to discover and register remote endpoints (e.g., BGP routers, virtual machines, etc), and negotiate network resources to provision and virtualize multiple virtual links to construct an overlay network for groups of users. The provisioning service should allow creating, modifying, and updating network tunnels between the discovered endpoint devices.

A key requirement is that these applications need a mediator layer/service to coordinate the communication between applications in virtualized servers or tenants and their underlying network to provision their QoS requirements. Subsequently, these requirements should be communicated to a control service that takes the configuration from the mediator service to provision and virtualize the underlying network resources. Additionally, ISPs need to constantly monitor the SLA (Service Level Agreement) conditions with remote applications and adjust the network resources if necessary. Therefore, a monitoring service is required to gather network topology information at a higher level of abstraction, and detect and report network failures.

2.2 Use Case 2: Internet Exchange Point

A Software-defined Internet Exchange (SDX) [8] is a large layer 2 domain that interconnects participant router peers of multiple ISPs together using the BGP routing protocol. It offers capabilities to address inter-domain routing issues by creating virtualized connections between routing endpoints. In the traditional approach, each BGP router is required to "learn" by itself the presence of other routers, select the best path for each prefix and re-advertise the new selected path. Additionally, a given router needs to control multiple switches at once and advertise its data simultaneously to multiple ISPs networks. These data need to be queued and made available for joining participants. For example, a router maybe be interested in a subset of traffic that conveys web HTTP traffic to its clients, or a subscribing router may be interested in video streaming traffic for its end users.

SDX may benefit from using the more expressive, proactive overlay SDN policies than conventional hop-by-hop destination based forwarding. SDX may also benefit from the data-centric, pub/sub for disseminating data only based on subscribers' interests.

2.3 Solution Requirements

Based on the real-world scenarios above, we derive the following requirements that must be met by POSEIDON:

1. Since the underlying network needs to be dynamically reconfigured, there is a need for the control plane to be notified whenever there is a change in the underlying physical network. This motivates the need for a publish/subscribe-based solution.

2. It is also important to provide the users with a generic policy framework that hides the heterogeneity in the underlying network devices. Thus, it is desirable to have a common interface and messaging capability between the control plane and forwarding plane where new devices can be added seamlessly.

3. It is important to provide the user with a flexible and higher-level of abstraction to program the system that can handle QoS issues and policies.

4. It is important for each instance of the virtualized overlay network instance to run in isolation from other networks, and to allow dynamically scaling up and down the number of applications/users in the virtualized data center. This motivates the creation of tunnels at the endpoint so that users/applications can be placed or migrated seamlessly without impacting the network performance.

3. POSEIDON ARCHITECTURE

This section details the POSEIDON architecture explaining how it meets the solution requirements of Section 2.3. The fundamental contribution of POSEIDON lies in how it realizes the proactive overlay SDN idea using data-centric pub/sub middleware. The use of data-centric, topic-based pub/sub is critical to our design because information can be represented as topics that can be easily queried using a variety of filter expressions, and can be disseminated to only interested entities thereby improving scalability.

To be compliant with the SDN architecture of separating the control plane from the data plane, and to not pollute it with extra infrastructure elements such as brokers, POSEIDON supports only two key elements as illustrated in Figure 1: a POSEIDON agent that resides in the control plane, and a routing agent that resides in the forwarding plane. The former relies on a policy management service in the SDN controller which supports intuitive and expressive interfaces at a higher-level of abstraction for applications to use. The latter receives these higher-level policy descriptions and translates them to low-level configuration commands in the switch thereby completely relieving the applications and developers from low-level programming.

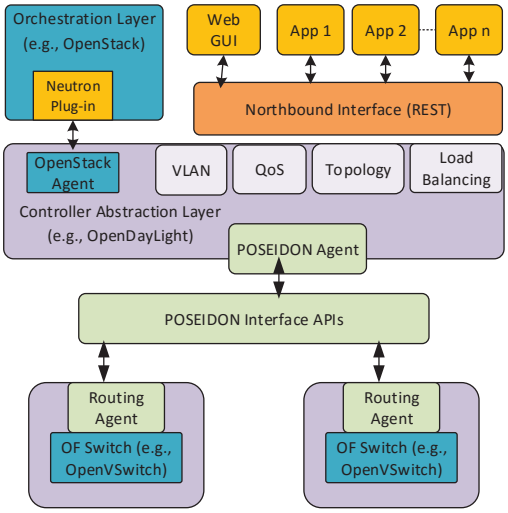

Figure 1: POSEIDON Architecture

POSEIDON provides a southbound interface based on the data-centric, pub/sub paradigm for communication between the two types of agents with a well-defined messaging protocol and message types. POSEIDON ensures no disruption to

existing OpenFlow-based SDN deployments since it encapsulates the original OpenFlow messages within its messages that are exchanged between the POSEIDON agent and routing agents. POSEIDON also allows new network devices to be introduced into the system without any disruption to the currently operational system. This feature is derived from the pub/sub system's discovery service, which promotes interoperability between endpoints so that remote participants can (i) dynamically learn about each other by sending participant declaration messages; and (ii) exchange information (e.g., QoS, data types, etc.) to match each other by sending pub/sub declarations. Consequently, POSEIDON provides more flexibility and programmability to users.

Intuitively, the POSEIDON middleware can manage one or more interconnected network slices, where each network slice is composed of a set of configurable SDN switches. Two or more neighboring network slices are interconnected inside a virtual global data space (GDS). Multiple GDSs are then connected to each other through edge SDN switches. POSEIDON exploits the expressiveness of data provided by the topics. Each topic is associated with the publication of data between each module. The content of each topic includes internal fields of our pub/sub routing services.

The rest of this section delves into the details of POSEIDON explaining the rationale for the design decisions.

3.1 POSEIDON Agent: The Control Plane

Figure 2 shows the internals of the POSEIDON agent, which comprises four types of nodes: monitor node, policy node, configuration node and control node. The monitor node collects information related to the health of the system including faults, errors, etc. The policy node includes a policy decision point and policy enforcement point both connected to a policy data store. The configuration node communicates with the orchestration layer via REST APIs to the applications. It also communicates with other configuration nodes through the distributed synchronization services provided by the built-in discovery mechanisms of the pub/sub system and with the policy node through a request-response model. The control node is used to define the rules for the proactive overlay. We now describe each of these node types in detail providing a rationale for their need in our design.

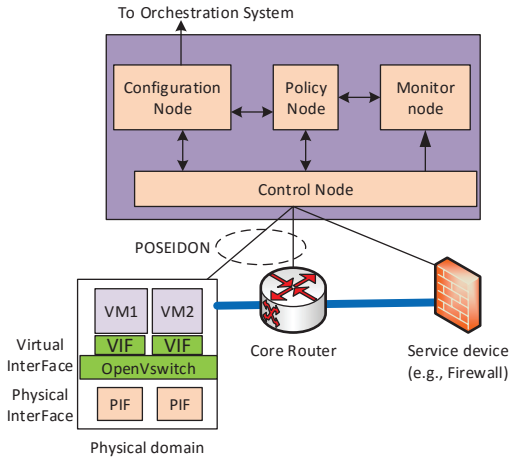

Figure 2: Internal components of the POSEIDON Agent

3.1.1 Configuration Node

The configuration node serves as the link to the cloud orchestration and automation systems using RESTful interfaces, which offer loosely coupled communication and high scalability. RESTful interfaces are used by cloud operators/applications to install configuration states to perform traffic engineering, e.g., load balancing. The configuration node can communicate with both the policy node and control node. For instance, the configuration node can query the policy node for the network virtualization policy used in the orchestration system. Similarly, it can direct the control node to install policy decisions in the network device. The configuration node uses topic-based filters provided by the pub/sub system for these communications. Topic-based filters are important because queries should return only pertinent information thereby reducing control-level traffic.

3.1.2 Policy control

Recall that network switches must be programmed to forward incoming traffic based on some rules specified by the controller. Our goal is to reduce the frequent and potentially non-scalable communication between the SDN controller and switches. Such forwarding rules and network management functions can be driven by different policies.

Figure 3 depicts the policy management service inside the policy node. This service provides policy control functions to the configuration node for ensuring best resource orchestration. To that end, the policy node infers policy decisions from application requirements specified through the configuration node and communicates them to the control node for enforcement. The policy module specifies how packet forwarding/updating abstractions should be realized.

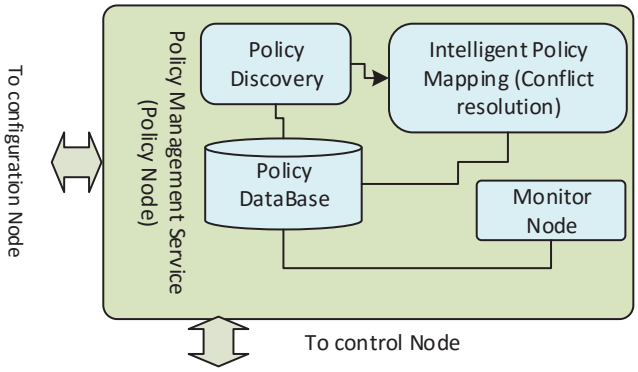

Figure 3: Policy Service within POSEIDON Policy Node

The core of the abstract set of policies we use is defined in [6] and [21]. The policies include a set of primitive actions, predicates, query policies, policy combinators and compositions. The primitives can be mixed to create complex policy functions to handle packets. The policy control functions ensure that the POSEIDON agent can continue to honor guaranteed service delivery to end-users. The policies also determine how network resources are allocated as well as how individual subscribers can take into account network flow control and application-oriented flow.

These policy abstractions are defined using the Pyretic description language [5], which enables users to express conflict resolution by composing different policies into a single set of matching rules in the network devices. Hence, rather

than supporting many policies, the abstraction policy layer combines policies from multiple participants to generate a set of rules that can be used for route advertisements without flooding the flow tables.

Pyretic provides a higher-level of abstraction to the operators, and automates the mapping of these higher-level descriptions into topic filter expressions used for querying. The policy database shown in Figure 3 is a repository of multiple abstract policies that can be composed using multiple parallel and sequential composition operators [25]. For instance, the sequential composition ($\gg$) redirects the output of one policy to the input of another forming a pipeline of sequential policies. Likewise, the parallel composition operator (+) applies multiple policy functions to the same packet and combines the results.

3.1.3 *Control Node*

The control node is responsible for installing the rules in the switches. It receives configuration states from the configuration node using content-based filtered topics, which contain the forwarding rules to be sent to the routing agent for installing new rules in the forwarding plane (i.e., data plane). Additionally, it exchanges network route information with other distributed control nodes and routing agents using the POSEIDON protocol. It also sends forwarding policies into the policy database for enabling intelligent decision making without having to contact the controller every time a new flow arrives. The control node holds a global snapshot of the network and the link states which helps it in performing the best route selection. Hence, the controller can configure virtual overlay tunnels based on the high-level information provided by the configuration node and the policy management service inside the policy node of the POSEIDON agent. To that end, the control node uses three types of topic-based messages for communication with the switches. The messaging protocol is described in Section 3.3.

3.1.4 *Monitoring, Analysis and Troubleshooting*

Monitoring the health of the system is critical. Figure 4 shows the internal structure of the monitor node. A monitor node supervises the network resources and communicates with applications using northbound REST interfaces. These interfaces are used to query statistics and analysis, and retrieve network operational states using the POSEIDON agent. Thus, the POSEIDON agent performs flexible and robust monitoring built atop the pub/sub model.

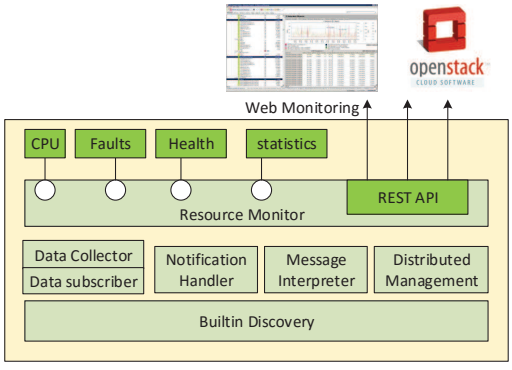

Figure 4: Monitoring node

Furthermore, the monitor node contacts the control node using the distributed management module to collect data from the network. The monitor node uses a data collector to communicate with the policy management module through the content-filter interface. That is, the monitoring service uses filter expressions to select data samples of interest by using SQL-like expression to store the information in the policy database.

3.2 Routing Agent: The Data Plane

As depicted in Figure 5, the routing agent is a user space process available on each instance of an OpenFlow-capable switch and acts as a local translator at the forwarding plane. It is responsible for exchanging control states between the POSEIDON Agent and the network devices using the POSEIDON protocol.

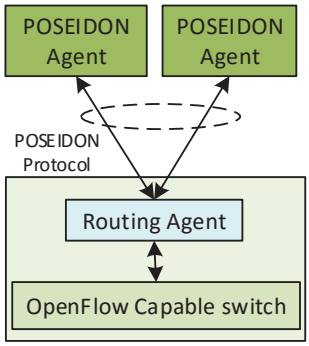

Figure 5: Internal view of the routing agent inside OpenFlow capable switch

The routing agent receives high-level configuration information from the POSEIDON agents and translates them to low-level details (i.e., QoS, VLAN tags, etc.), which is understood by the switches in the overlay. To support a uniform abstraction that can cover a range of these low-level possibilities, POSEIDON supports the notion of a built-in data isolation called *Partition*, which is a scoping mechanism to organize traffic by creating non-overlapping logical virtual data spaces for dynamic association of participants. Partitioning the network into several network slices can increase the scalability of the system by allowing multiple advertisements and subscriptions.

For virtual LANs (VLANs), *Partitions* can be seen as VLAN identifiers (i.e., VLAN tags) so that routing agents belonging to the same physical infrastructure can easily be isolated into different VLANs. These partitions are mapped into specific VLAN ports for grouping the sent data into specific receivers. Consequently, the routing agent translates these partitions into VLAN tags and installs the forwarding states within the OpenFlow entries of the SDN switches.

Likewise, a partition can be translated to an IP multicast address to perform multicast VLAN registration to efficiently distribute data across switches thereby reducing the bandwidth consumed by the multicast traffic. In such a case, POSEIDON can check whether routing agents belonging to a specific partition are still alive. The pub/sub discovery process reports whether new virtual switches have joined or left the network. Besides, the routing agent carries and reports analytics, health status, errors, and statistics from the forwarding plane to the distributed POSEIDON agents.

3.3 POSEIDON Messaging Protocol

In this section we describe the three types of messages currently supported by the POSEIDON messaging protocol: peer discovery, network configuration, and monitoring messages, which are exchanged between the control node of the POSEIDON agent and the routing agents.

3.3.1 Endpoints Discovery Message

POSEIDON relies on the underlying pub/sub system's built-in discovery protocol to identify the tunnel endpoints matching POSEIDON agents and routing agents. Discovery messages are sent periodically (i.e., regulated by a heartbeat) to check the liveliness of different POSEIDON peers. Listing 1 shows the peer discovery message exchanged between POSEIDON agents and routing agents. The "sender" field describes the IP address of the publishing endpoint. It can also contain the MAC address in the context of a layer 2 participant. The "sender_ID" describes the "VLAN ID" of a participant or any other membership information that can uniquely identify a participant within its group. The format is extensible and customizable to the underlying pub/sub system.

Listing 1: Built-in participant Discovery

```
<?xml version="1.0"?>
<-- Discovery topic -->
<struct name="discovery">
<member name="sender" type="string" key="true"/>
<member name="sender_ID" type="string" />
<member name="receiver" type="string"/>
<!-- QoS policies ... -->
<member name="LivelinessQoS" type="string"/>
<member name="DurabilityQoS" type="string"/>
<member name="TopicDataQoS" type="string"/>
<member name="GroupDataQoS" type="string"/>
<!-- more QoS policies ... -->
<!-- Session Management -->
<member name="scope_announce" type="sequence">
<member name="register_context" type="sequence">
<member name="session" type="sequence" key="true"/>
</struct>
```

For example, depending on the underlying pub/sub system's features, the discovery message supports reconfigurable QoS policies to control many reliability aspects, such as liveliness. For instance, when using the OMG Data Distribution Service (DDS) pub/sub technology, in Listing 1, "*LivelinessQoS*" is used to ensure automatic discovery and liveness check of endpoints. The endpoint discovery process uses inbound session management to ensure that communication will be established end-to-end. The field "scope_announce" in Listing 1 describes the announcement of a session from a participant to all others. The "scope_announce" is a sequence of fields that includes the forwarding path, the session heartbeat, and discovery data.

3.3.2 Network Configuration Operation

Once the endpoints are discovered, it is important for the POSEIDON agent to configure all the identified routing agents with flow rules. Listing 2 illustrates a configuration message used by the POSEIDON agent to populate the Routing Information Base (RIB) of the routing agent.

Listing 2: Configure BGP Path

```
<?xml version="1.0"?>
<struct name="bgp_advertise">
<member name="sender" type="string" key="true" />
<member name="receiver" type="string"/>
<member name="sender_id" type="long"/>
<member name= "node" type="string" />
<member name="instance-id" type="long">
<--! BGP Network Layer Reachability
Information (NLRI) -->
<member name="nlri" type ="string">
<!-- Label for label switch routers (LSRs) -->
<member name="label" type="long">
</struct>
```

3.3.3 Monitoring messages

Currently, POSEIDON supports network management and monitoring capabilities through the monitoring message. Listing 3 shows the topic data used by POSEIDON to query the state of a SDN switch. This topic message contains information about the fault, statistics, errors, and health of the network device. It also includes information about the sender of that request (i.e., IP address, VLAN tag), the entity which sent the report and other information including statistics, faults, and errors, etc.

Listing 3: Query Network Statistics

```
<?xml version="1.0"?>
<struct name="report_sate">
<member name="sender" type="string" key="true" />
<member name="sender_id" type="long"/>
<member name="receiver" type="string"/>
<member name= "node" type="string" />
<member name="statistics" type="struct" >
<member name="faults" type="struct">
<member name="health" type="struct">
</struct>
```

4. VALIDATING THE POSEIDON APPROACH FOR REAL-WORLD USE CASES

The aim of this section is to validate the POSEIDON approach for the same use cases we used to motivate the work.

4.1 POSEIDON for Interconnecting Distributed Cloud Networks

Figure 6 illustrates how POSEIDON can be used to scalably interconnect distributed cloud networks. In contrast to the traditional approach that uses physical links (shown as PL_I, i = 1..5 in Figure 6) to exchange routing information, POSEIDON provides a topology abstraction to simplify interconnection by virtue of creating a big virtual switch "v_sw". This is feasible because the abstracted physical topology can be ported to a collection of distributed routers due primarily to the policy repository (Section 3.1.2). Recall that this repository holds a topology abstraction library that determines whether to aggregate multiple underlying switches into a single overlay or splitting a physical node into several virtual ones.

Listing 4 using the expressive description in Pyretic shows the topology transformation of the underlying network where the physical switches "*SW1, SW2, and SW3*" are aggregated to create the virtual switch "v_sw" in the overlay network.

Listing 4: Mapping the Physical devices to Virtual ones

```
def virtual_switch(topo):
```

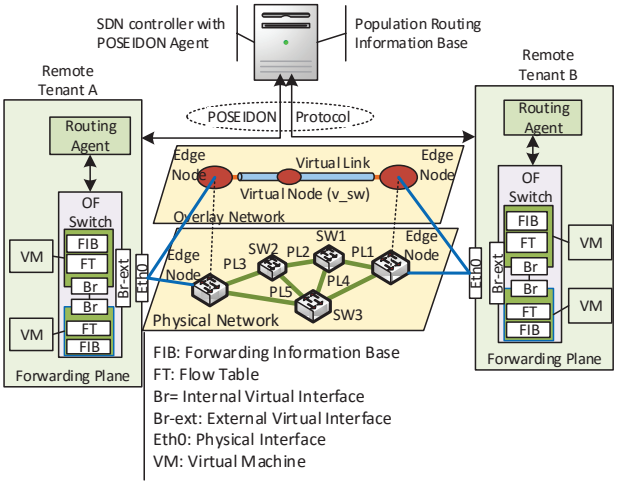

Figure 6: POSEIDON for Interconnecting Multiple Data Centers over the Internet

```
    self.v_sw = 1
    self.vport = 1
    for (sw, port) in topo.egress_location:
        vmap[(v_sw, vport)] = (sw, port)
        vport +=1
    return vmap
```

The next step consists of creating the transformation policy which allows virtualizing the underlying network. To generate the topology abstraction policy that abstracts the underlying switches into the virtual switches, the *virtualize()* function in Listing 5 takes SW1, SW2, and SW3 as input and renders "v_sw" as a topology transformation. The *merge* abstracted policy takes a list of the underlying physical switches and renders a single overlay virtual switch "v_sw". It selects the shortest path to forward packets from one edge router to another. Thereafter, the routing agent at the customer edge router implements translation of the *virtualize* policy into a single new policy for the underlying router.

Listing 5: Virtualize topology abstraction function

```
virtualize(v_sw,
           merge(name=3,
           from_switches=[sw1, sw2, sw3]))
```

This abstracted policy creates a virtual tunnel in the overlay network to allow distributing the route's information among autonomous systems and enable a high level of aggregation between customer interfaces and provider's edge devices. Each customer device implements a POSEIDON Agent to translate the configuration sent by the control plane through the POSEIDON protocol.

4.2 POSEIDON for Internet Exchange Point

Figure 7 depicts how POSEIDON uses the BGP protocol for simplifying the exchange of network reachability information using a POSEIDON agent in the SDX controller. Unlike traditional approaches where each BGP router should "learn" by itself the other routers, our approach acts as a route reflector that learns BGP route advertisements, selects the best path for each prefix, and re-advertises the new selected path on the appropriate session using the *scope_announce* field in the endpoint discovery message described in Listing 1.

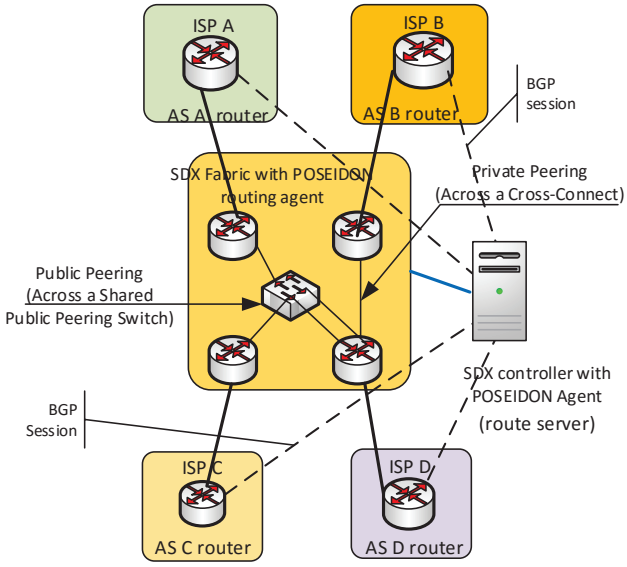

Figure 7: POSEIDON for Software Defined Internet Exchange

The POSEIDON agent runs an SDN application to drop, modify, update, and forward traffic among different Autonomous Systems (ASs). In particular, it combines BGP reachability information from all AS' edge routers into a single set of policies to give each router the illusion as if it has its own virtual switch, while ensuring the isolation of the traffic between different BGP participant routers as illustrated in Figure 8.

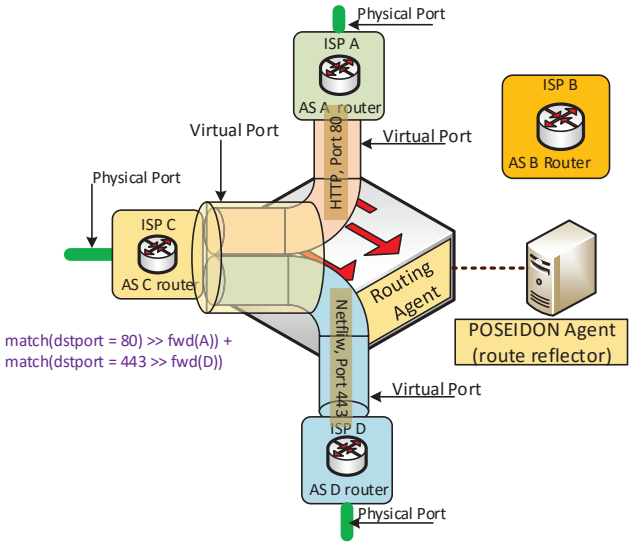

Figure 8: Transforming the underlying topology into single virtual switch

ISP C has virtual interfaces to connect to virtual interfaces of ISP A and ISP D through the switch. Listing 6 depicts the parallel composition of the abstracted policy that allows

ISP C to filter HTTP traffic towards ISP A and forwards Netflix video streaming on UDP port 443 towards ISP D.

Listing 6: Forwarding Policy in ISP C

```
match(dstport = 80) >> fwd(A)) +
match(dstport = 443 >> fwd(D))
```

The policies include conflict resolution rules to program the virtualized switch and avoid the policies' interference conflicts between edge routers. Additionally, the POSEIDON agent implements a router reflector to replace the traditional BGP route server [12]. These routing decisions are based on per-endpoint policy control and the best-path calculation for each set of paths available between the distributed routes.

The route reflector is an SDN application at the control plane that checks the advertised routes from each AS and performs path filtering at that exchange endpoint. It also allows enforcing the traffic ratio among different routing endpoints. POSEIDON splits the virtual tunnel of AS C into two small virtual tunnels, each conveying the filtered traffic for each AS separately as shown in Figure 8. Consequently, each network provider can apply a customized route selection process to select one or more best routes to each Internet destination.

In traditional approaches, each BGP speaker should be configured separately to provide point-to-point communication with all the other speakers. In contrast, in our approach the big virtual switch in Figure 8 hides the complexity of configuring each BGP speaker separately. The virtualization policy supported by our POSEIDON agent creates virtual tunnels between edge routers. It creates the overlay big switch by virtualizing the underlying network, i.e. the physical BGP edge routers. The resulting virtual overlay switch maps the ingress physical ports into virtual ports to create the illusion that each router has its own switch interface.

Furthermore, the control plane translates the local RIB into the corresponding prefix in each nearest BGP speaker to forward multiple paths for the same speaker on a single BGP session. Path adding/removing is negotiated bidirectionally between the POSEIDON Agents and the service provider routers. The POSEIDON agent ensures that there is no inactive, invalid or even suboptimal path to its peering routers.

5. PERFORMANCE EVALUATION

This section describes results of experimental evaluation of POSEIDON. Specifically, we describe the results for different network metrics such as bandwidth, latency, network overhead, and scalability.

5.1 Experimental Setup

POSEIDON has been evaluated for the two use cases described in Section 4. The experiments were conducted on a SDN testbed running the Mininet SDN tool [17]. Mininet is a Linux-based lightweight SDN virtualization tool for running large number of OpenFlow network equipment, and allows experimenting with different types of network topologies. We also integrated the different modules comprising our solution into the NOX [7] SDN controller, which is written in C++ and Python programming languages. We implemented the reactive OpenFlow in the SDN controller so that each time a new packet arrives at the edge router, it sends a request using PACKET_IN message to the controller. The controller examines the packet header of the incoming packets and checks whether a new entry should be created and new actions should be applied to those packets. Then, the controller sends PACKET_OUT message to all the underlying routers from the source to the destination. Additionally, to implement the proactive approach, we used the OMG Data Distribution Service (DDS) as the underlying topic-based, data-centric pub/sub middleware. Thus, POSEIDON can create virtual tunnels between edge routers. To install new rules for establishing the tunnel, the controller sends OpenFlow messages to the tunnel endpoints, i.e. the edge routers of that tunnel.

5.2 Evaluating the End-to-end Latency

Rationale. End-to-end latency is critical to many SDN-enabled applications since it allows routing traffic in predictable time scales. To evaluate the timing performance of our solution, we consider the end-to-end delay as the time duration to send a packet from source to destination. The measurement of the one way delay is not straightforward because packets experience different network delays including processing delay, queuing delay, transmission and propagation delays. Thus, we have considered the two-way latency, i.e., the Round Trip Time (RTT). We then consider one-way delay as half of *RTT*. Furthermore, to evaluate the effectiveness of our approach, we compared the latency results with those of the traditional, reactive hop-by-hop OpenFlow data dissemination.[1]

Analysis. Figure 9 depicts the end-to-end latency for both the reactive OpenFlow and our proactive overlay SDN approach. A close inspection of Figure 9 shows that the latency experienced by our proactive overlay solution is close to 20 milliseconds on average (reaching 35 msec in worst case), which is better than the reactive approach that showed worst-case latency up to 75 milliseconds. Indeed, the reactive OpenFlow is constantly changing in reaction to the

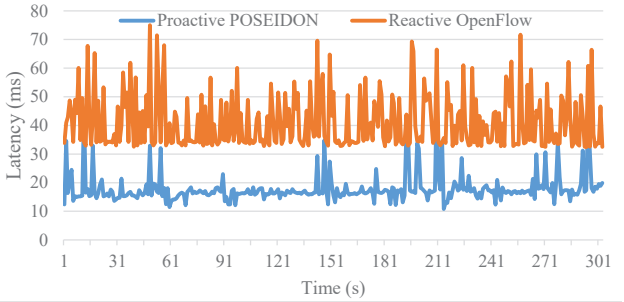

Figure 9: Latency: Proactive POSEIDON vs. Reactive OpenFlow

current network conditions. The volatile reactivity to the changing events introduces high rate of changes because reactive OpenFlow must continually update all the switches along routing paths. Moreover, the OpenFlow agents in every switch should lookup in their flow tables to check whether a matching rule/action for a given packet flow can be found. If no matching is found in the flow entries, the OpenFlow agents should send requests, i.e. *PACKET_IN*

[1]Recall that the pure proactive approach is impractical and mostly of theoretical value.

messages, to the controller to create new flow entries for that given flow. Accordingly, reactive OpenFlow experiences larger latencies and substantial jitter than our proactive solution, which attempts to address the latency concerns before it becomes a problem in the network.

As seen, our overlay routing paths do not introduce extra processing latency in the overlay network and demonstrate less jitter. Rather, our approach creates a virtual tunnel between two end switches along the routing path so that packet processing is enabled only at the beginning and the end of the tunnel. Our solution improves the network latency by enabling proactive overlay SDN anonymity to all applications running on top of overlay routing protocols. Our approach preserves the higher performance characteristics required over the overlay routing path while hiding the complexity of the underlying network topology.

5.3 Evaluating Bandwidth usage

Rationale. Since SDN networks must share the same physical infrastructure with other existing applications, the bandwidth usage is a concern. Thus, data dissemination in POSEIDON middleware should be protected against any network fluctuation such as congestion. To evaluate the usage of the shared links, we consider each application generating best effort traffic close to 10 MB/sec. Our aim is to evaluate whether our approach shows better results to protect the flow in the face of bandwidth fluctuation. We also compared our approach with the reactive OpenFlow solution.

Analysis. Figure 10 illustrates the bandwidth usage for both the POSEIDON approach and the reactive OpenFlow. Both approaches allow sending data at 10 MB/s. However, reactive OpenFlow experiences several bandwidth limitations and the traffic becomes irregular. The bandwidth decreases drastically at different time intervals. The packet flow experiences several packet losses. In contrast, in our proactive solution, the throughput decreased only at the initialization phase comprising the creation of the tunnel, i.e., establishing the connection between two end switches across the overlay routing path. Thereafter, the bandwidth becomes regular and protected against the fluctuation during the rest of the experiment. Accordingly, the results show that our solution is successfully able to overcome the limitations of the reactive OpenFlow approach by providing better usage of network bandwidth. This is because the POSEIDON agent at the controller receives fewer request messages from the data plane since the virtual path is already setup inside the virtual tunnel.

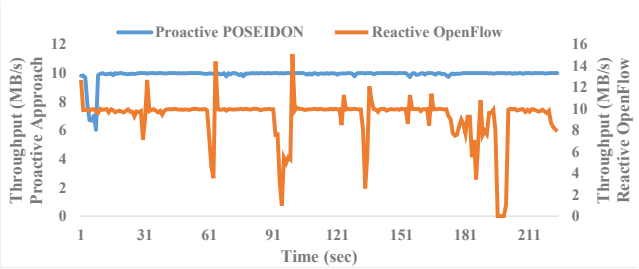

Figure 10: Efficient Bandwidth usage for POSEIDON

5.4 Evaluating POSEIDON Scalability

Rationale. The design choice of our proactive SDN solution is similar to distributed systems as it was designed with scalability concerns in mind to allow its deployment in large-scale SDN scenarios. Therefore, to evaluate the scalability of our proposed solution, we consider the average response time in processing the network requests. Specifically, we increased the number of the data plane nodes and we evaluate the time latency for every additional introduced node.

Analysis. Figure 11 depicts the latency of the proposed proactive SDN approach as a function of the number of data plane equipment across the path. A close inspection of the figure reveals that the average latency of our proposed solution is close to 30 ms, while the maximum value of this delay remains close 50 ms.

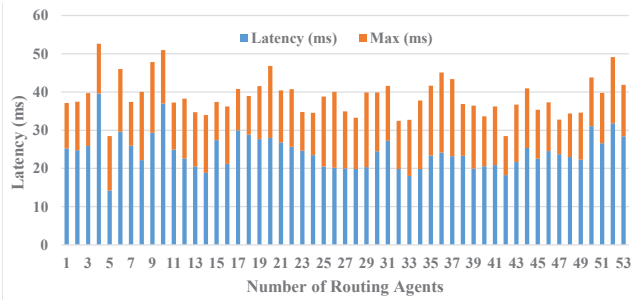

Figure 11: Evaluating the Scalability

Since our proactive approach creates a virtual tunnel between end points, increasing the number of core routers across the data path will not necessarily linearly impact the performance of the network. There are a couple of reasons for this: first, core routers are usually high performance network equipment that forward incoming packets without any additional processing; second, the core routers remain transparent to the network as the virtual tunnel will cross them seamlessly.

5.5 Evaluating Network Overhead

Rationale. As POSEIDON is used for interconnecting data centers, there are several critical metrics that should be considered in evaluating the overhead introduced by POSEIDON. First, the CPU load is usually impacted by the control modules such as the configuration module to perform connection setup and tear down. Moreover, the forwarding operations performed by the POSEIDON routing agent can come at a cost of increased memory utilization in the overlay switches, which is a critical issue in data center networks. In particular, memory and queuing buffers hold packets and connection state information of the traffic transiting across the switch. Therefore, there is a need to maintain the memory utilization as low as possible to avoid buffer overflow.

Analysis. Figure 12 illustrates the control plane overhead for both the reactive OpenFlow approach and our proactive solution. The figure shows that the control traffic of the former varies between 5 Kbits/sec and 15 Kbits/sec, while the network overhead of the proposed proactive approach is less than 5 Kbits/sec most of the time and increases a little up to 5 Kbits/sec at regular time intervals. Indeed, the reactive OpenFlow requires processing several network requests for network state update, topology discovery, failure recovery, etc. Unlike the reactive approach, our proactive overlay

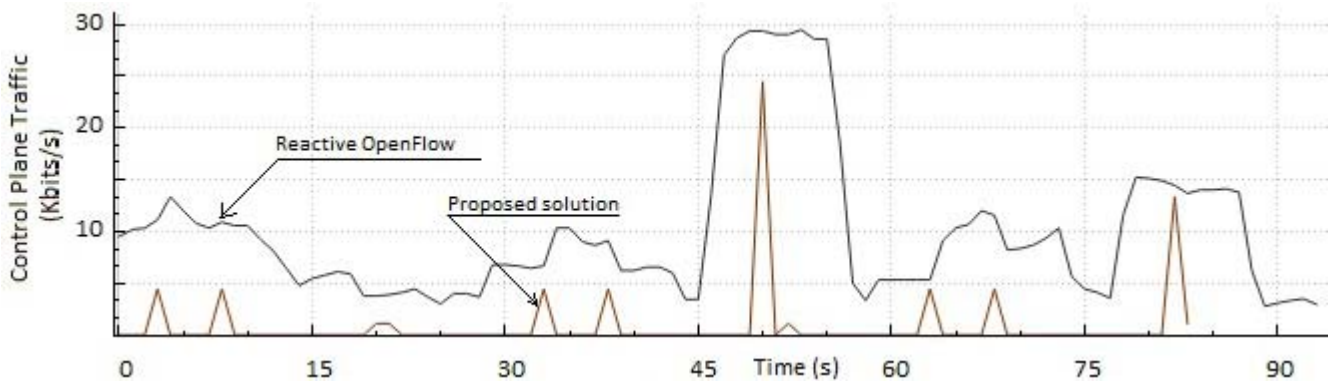

Figure 12: Control Plane Overhead: Reactive OpenFlow vs. Proactive POSEIDON

solution sends *Endpoints Discovery Message* that acts like heartbeat messages at heartbeat periods to discover whether new nodes have joined or left the network. As discussed in Section 3.3, the discovery messages are sent periodically to check the liveliness of different peers. In particular, the *LivelinessQoS* and the *scope_announce* parameters ensure automatic discovery of end points as well as heartbeat periodic data discovery. Thus, the evaluation of the network overhead confirms our claims on providing better network performance compared to the traditional reactive approach.

6. RELATED WORK

In this section we survey relevant research efforts and compare them to POSEIDON. Our SDN-based literature survey indicates that there does not exist a clearly identified role for middleware platforms (e.g., whether it should be used in the northbound or southbound or east-west interfaces) with the aim of interconnecting applications to SDN network and providing the freedom to the developers to define the way they would use it in SDN. There have been some efforts, however, to introduce middleware in SDN.

The authors in [24] introduce a broker-based framework called DISCO as an east-west interface to coordinate federated SDN controllers using RabbitMQ [11] middleware over the Advanced Message Queuing Protocol (AMQP) [31]. DISCO allows the brokers to make routing decisions rather than letting the SDN network application perform this task. As a broker-based solution, DISCO routes the control traffic between distributed controllers, however, it does not perform any routing decisions at the SDN forwarding plane, which is left to the control plane applications. In contrast, POSEIDON supports self-formed federation that can perform data dissemination/advertisement over the global data space of topics. As such, POSEIDON operates as a repository federation in which individual repositories can participate in a global federation in a fully distributed manner.

Authors in [9] proposed a data-centric IoT architecture in which a pub/sub middleware is integrated into SDN as a northbound interface for enabling agile and flexible network orchestration. Similarly, authors in [14] describe a pub/sub middleware called MIDAS that leverages OpenFlow to control the resource management strategy and low-level switching configurations. Unlike these approaches that consider the middleware layer as a northbound interface, POSEIDON can be considered as a southbound interface to connect distributed overlay SDN data planes.

Likewise, authors in [3] introduce an OpenFlow-enabled middleware for virtual machine migration in data centers at the SDN control plane. On the other hand, POSEIDON defines both the control plane functionality as well the data plane forwarding routines for supporting proactive SDN model. It also describes the messaging protocol to connect them together.

Authors in [15] propose a pub/sub architecture for SDN where the controller establishes line-rate content-based matching semantics to disseminate routing information to SDN-enabled switches. The common shortcoming of content-based systems is their dependency on the application layer to perform pub/sub operations. They also do not map directly to multicast communication unless a routing engine is added for group communication, which may result in higher bandwidth consumption and higher latency [30]. POSEIDON natively supports many-to-many communication pattern and can provide reliable group communication for scalable one-to-many and many-to-many data distribution.

Recent efforts have already showed how pub/sub and SDN can be combined to perform content filtering at the level of SDN controllers. For example, LIPSIN [13] uses Bloom filters in data packets to infer the underlay topology from the overlay network, and enable efficient multicast communication. Despite this, LIPSIN requires substantial architectural changes for moving end-point oriented systems to their architecture. Similarly, PLEROMA [29] uses content-based filters in SDN to enable group communication as well as allows running multiple virtual overlay domains. Likewise, authors in [1] introduced an event-based middleware to offer distributed SDN control plane management and configuration. Unlike these works, our contribution provides a distributed pub/sub service in the control plane along with a routing agent for enabling proactive overlay data plane communication.

Recently some efforts have explored the use of Extensible Messaging and Presence Protocol (XMPP) service as an alternative to OpenFlow in hybrid SDN networks [18]. The XMPP middleware is used to distribute control plane and management plane information to end servers that serve to enhance the communication between data centers in overlay networks and physical devices in the underlying network. The disadvantages of XMPP and its related technologies (i.e., roster, presence and routing functions) exist in differ-

ent contexts. In particular, XMPP is considered to be a standard only for the wire protocol, i.e., XMPP is agnostic in relation to the data being transferred. XMPP is also stateful, which makes it more difficult to scale because each server needs to know the entire state in order to serve a request. Typically, the XMPP clients and servers utilize the domain name system (DNS) to resolve a server's domain name into an address they can connect to. XMPP also requires centralized services to exchange messages between server-to-server federation, which makes it inefficient in duplicating messages when distributing them to multiple destinations. This is where utilizing POSEIDON for message distribution is more beneficial than XMPP. POSEIDON uses the built-in pub/sub discovery service to allow publishers and subscribers to dynamically and continuously discover each other without the need to contact any name servers.

The OpFlex [28] control protocol is introduced to configure and monitor all connected devices. The OpFlex protocol is founded on the concepts of declarative policy-driven system to control and program a large set of physical and virtual network devices. OpFlex is a request-response protocol based on JSON-RPC [22], where each component sends a request to query the information from its peer element. However, there are several disadvantages of RPC with respect to message passing since it may incur a severe degradation in performance due to marshaling/unmarshaling of messages (i.e., context switching increases scheduling costs) and may have to deal with added complexity in configuration for simple scenarios.

Unlike these approaches, POSEIDON is fully distributed so there is no single point of failure in the network during the communication. It also supports reconfigurable resource management policies for efficient use of the bandwidth, network and memory resources. Additionally, it supports the proactive overlay SDN, which makes it a suitable technology to ensure scalability, reliability, and flexibility.

7. CONCLUSION

Proactive overlay software defined networking (SDN) aims at overcoming the scalability limitations of reactive hop-by-hop approaches for large-scale networks and the practical limitations of the pure proactive approach. Although both approaches are supported by the dominant SDN technology OpenFlow, its APIs are too low-level and lacks intuitive abstractions to build reusable SDN applications. To overcome these limitations, in this paper we described the POSEIDON middleware for realizing proactive overlay SDN. POSEIDON proposes an efficient matching of advertisement and subscription using the data-centric publish/subscribe (pub/sub) paradigm to support flexible and programmable SDN networks. The paper demonstrates how POSEIDON can be used in two real-world use cases. The experimental evaluation of POSEIDON validates our claims about (1) the ease of configuring the overlay SDN forwarding tables of the data plane directly through the middleware using a software control layer inside the SDN controller, and (2) its scalability properties.

The following lessons were learned conducting this research, which illustrates the limitations of the work while also highlighting opportunities for further research in this area:

- **Applying POSEIDON solution for IoT systems.** The modular architecture of POSEIDON makes it suitable for large number of wired networks such as cloud environment, Cloudlets, and multi-tenant networks. However, there are many reasons to extend POSEIDON to support Internet of Things (IoT) systems. To enable POSEIDON support for IoT, it should be extended to integrate new protocols such as 6LowPAN [16] and ROLL [32]. Meanwhile its footprint, stemming from its reliance on a pub/sub middleware, should be reduced to suit the embedded, resource-constrained systems. We also believe that such an approach will draw attention of the research community to the need to bring agility and programmability of SDN for scalable and high performance IoT networks.
- **Securing Flow Forwarding in POSEIDON.** POSEIDON has the ability to interconnect hundreds of SDN controllers and devices at different scales. Currently, POSEIDON supports built-in security policies to deliver safety required by the control plane. At the data plane, since the flow table entries are written as pieces of software modules, their content may be altered due to the increasing number of DDoS and malware attacks, and phishing activities. Therefore, we believe that sophisticated encryption and authentication mechanisms to overcome hackers, and recovery of packets from failure are needed. For example, role-based and policy-based access control [26] could be added to SDN to support data integrity, pedigree, confidentiality, and non-repudiation.

Acknowledgments

This work was funded by the Fullbright Visiting Scholars Program, NSF CNS US Ignite 1531079 and by AFOSR DDDAS FA9550-13-1-0227. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Fulbright, NSF and AFOSR.

8. REFERENCES

[1] S. Bhowmik, M. A. Tariq, B. Koldehofe, A. Kutzleb, and K. Rothermel. Distributed control plane for software-defined networks: A case study using event-based middleware. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, DEBS '15, pages 92–103, 2015.

[2] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 99–110, 2013.

[3] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui. Openflow supporting inter-domain virtual machine migration. In *Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on*, pages 1–7, May 2011.

[4] M. P. Fernandez. Comparing openflow controller paradigms scalability: Reactive and proactive. In *Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications*, AINA '13, pages 1009–1016, 2013.

[5] N. Foster, A. Guha, M. Reitblatt, A. Story, M. Freedman, N. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison. Languages for software-defined networks. *IEEE Comm Mag*, 51(2):128–134, February 2013.

[6] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. *SIGPLAN Not.*, 46(9):279–291, Sept. 2011.

[7] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3), July 2008.

[8] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett. Sdx: A software defined internet exchange. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 551–562, 2014.

[9] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif. Publish/subscribe-enabled software defined networking for efficient and scalable iot communications. *Communications Magazine, IEEE*, 53(9):48–54, 2015.

[10] J. H. Han, P. Mundkur, C. Rotsos, G. Antichi, N. Dave, A. W. Moore, and P. G. Neumann. Blueswitch: enabling provably consistent configuration of network switches. In *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*, pages 17–27, 2015.

[11] V. Ionescu. The analysis of the performance of rabbitmq and activemq. In *RoEduNet International Conference - Networking in Education and Research (RoEduNet NER), 2015 14th*, pages 132–137, Sept 2015.

[12] E. Jasinska, N. Hilliard, R. Raszuk, and N. Bakker. Internet exchange bgp route server. Internet-Draft draft-ietf-idr-ix-bgp-route-server-09, IETF Secretariat, October 2015.

[13] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. Lipsin: Line speed publish/subscribe inter-networking. *SIGCOMM Comput. Commun. Rev.*, 39(4):195–206, Aug. 2009.

[14] A. King, S. Chen, and I. Lee. The middleware assurance substrate: Enabling strong real-time guarantees in open systems with openflow. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on*, pages 133–140, June 2014.

[15] B. Koldehofe, F. Dürr, M. A. Tariq, and K. Rothermel. The power of software-defined networking: Line-rate content-based routing using openflow. In *Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing*, MW4NG '12, pages 3:1–3:6, 2012.

[16] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational), aug 2007.

[17] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[18] P. Marques, L. Fang, P. Pan, A. Shukla, M. Napierala, and N. Bitar. Bgp-signed end-system ip/vpn, October 2015.

[19] P. Marques, L. Fang, N. Sheth, M. Napierala, and N. Bitar. Bgp-signaled end-system ip/vpns. IETF, Oct 2015.

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, and J. Rexford. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, pages 69–74, 2008.

[21] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing software-defined networks. In *10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[22] M. Morley. JSON-RPC 2.0 Specification, Mar 2010.

[23] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys Tutorials, IEEE*, 16(3):1617–1634, 2014.

[24] K. Phemius, M. Bouet, and J. Leguay. Disco: Distributed multi-domain sdn controllers. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–4, May 2014.

[25] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker. Modular sdn programming with pyretic. *USENIX, the Advanced Computing Systems Association*, 38(5), 2013.

[26] S. Scott-Hayward, S. Natarajan, and S. Sezer. A survey of security in software defined networks. *Communications Surveys Tutorials, IEEE*, 18(1):623–654, Firstquarter 2016.

[27] S. Shirali-Shahreza and Y. Ganjali. Rewiflow: Restricted wildcard openflow rules. *SIGCOMM Comput. Commun. Rev.*, 45(5), 2015.

[28] M. Smith, R. Adams, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher. Opflex control protocol, October 2015.

[29] M. A. Tariq, B. Koldehofe, S. Bhowmik, and K. Rothermel. Pleroma: A sdn-based high performance publish/subscribe middleware. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, pages 217–228, 2014.

[30] M. A. Tariq, B. Koldehofe, and K. Rothermel. Efficient content-based routing with network topology inference. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, 2013.

[31] S. Vinoski. Advanced message queuing protocol. *Internet Computing, IEEE*, 10(6):87–89, Nov 2006.

[32] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), Mar. 2012.