Software-Defined Network Virtualization with FlowN

Dmitry A. Drutskoy

Master's Thesis

In Partial Fullfillment of the Requirements
for the Master of Science in Engineering
Department of Computer Science
Princeton University

Adviser: Jennifer Rexford

June 2012

Abstract

Network virtualization enables multiple control planes to run concurrently and each affect only its portion of the data plane state and configuration. With the standardization of the interface between the control plane and data plane, software defined networking is a natural fit with network virtualization as any software can run to interact with the data plane. A virtualization layer can then be created which modifies the view of the physical network and enables multiple control planes to run concurrently – each targeting a different view of the dataplane. In contrast to previous virtualization solutions, which also attempt to abstract the network design, we propose that for shared infrastructures, such as cloud computing, the abstraction should be decoupled from the virtualization support, to support different abstractions for different tenants. With this, we present the FlowN system, which implements full network virtualization, with which each virtual network: (i) is independent and can control the entire flow address space, (ii) can control the entire path within the network, (iii) can be an arbitrary topology.

The FlowN architecture is designed to be scalable through two novel approaches in software defined networking (SDN): the use of existing database technology for mapping virtual to physical topologies, and a shared controller platform analogous to container-based virtualization. Our prototype extends the OpenFlow NOX controller and uses a MySQL database. We demonstrate the latency efficiency of our container based approach by comparing it to FlowVisor and unvirtualized NOX.

Acknowledgments

I would like to thank my adviser, Jennifer Rexford, for supporting and guiding me through this work as well as my time at Princeton University. Her support, both academically and as a person, have helped me through this time. I strongly believe that none of this would have been possible without her help. I would also like to thank Eric Keller, for his ideas for this work and continuous support of me even in the face of difficulties. Finally, I'd like to thank Princeton University for the amazing opportunity to study at this place and meet the above mentioned people, as well as great faculty and students, some of which will remain my friends for life.

Contents

1 Introduction

Software Defined Networking (SDN) offers unprecedented control to network administrators by running services on a logically-centralized controller that uses a standard API (such as OpenFlow [14]) to install packet-handling rules in the underlying switches. In addition to supporting new networked services, SDN naturally supports network virtualization, where the network can be abstracted to support many different controller applications that each are tailored to the needs of a particular "tenant" or class of traffic. Today's solutions couple the abstraction with the virtualization technology, such as proposing that a shared network infrastructure be presented to tenants as one "big switch" (as in the case of Nicira [17]). We believe that the abstraction presented to the tenants should be decoupled from the virtualization technology that enables tenants to share the physical infrastructure.

In this paper we present a full network virtualization solution. With full network virtualization, the virtualization layer presents arbitrary topologies to the SDN controller applications. The SDN applications are in full control over the entire flow space within that network and isolated from one another. This approach provides a general solution to network virtualization, where each tenant has control of the entire virtual network and can deploy any abstraction on top of the virtual network that they want. In this fashion, we can accommodate the ease of management of any of the current solutions, while allowing for potential fine-grained control of the network or other abstractions to be added.

As our system is designed to operate in a datacenter environment, we are concerned with being able to scale in physical network size and number of virtual networks, as well as operate virtual networks using the least amount of overhead, in both latency for the virtual networks and resource consumption for the machine housing the virtual network controllers. In this paper, we use two crucial design decisions to address these challenges.

The first challenge is efficiently providing a mapping between the virtual network resources and the physical resource they currently map to. Our approach is to use a database, which provides a natural fit due to the representation we chose for virtual networks, and allows us to capitalize on advances in database research to enable the virtual to physical mapping in an efficient and scalable manner.

As every tenant is running their controller application, we have the second challenge in efficiently running these applications alongside the controller software which interacts with the physical network. Current solutions, such as Flowvisor [18], act as a proxy which presents an interface to the tenant application which is the same as interacting with the physical netowrk. With this, each tenant must run its own controller software to interact with the virtual network switches. We opt for an approach inspired by operating system-level virtualization where the kernel of the operating system is shared amongst isolated containers. In our case, the controller software, which interacts with the physical switches, is shared amongst isolated tenant applications.

The remainder of the paper is organized as follows. In Section 2, we argue for the decoupling of the virtualization from the abstraction layer, and discuss the design requirements for the virtualization layer. In Section 3 we present the system architecture. In Section 4, we discuss the database design more in-depth. In Section 5, we present our prototype and evaluate it, and in Section 6 we discuss potential future work on this project. In Section 7, we discuss some of the related work that has been done in this field. We conclude in Section 8.

2 Virtualization and Abstraction

The recent availability of hosted cloud computing infrastructures has lowered the barrier for creating a new networked service. Likewise, experimental platforms such as GENI provide researchers with a platform to perform larger scale research experiments. A key enabling technology in these infrastructures is virtualization which enables sharing of physical resources. The prevalence of virtual server technology has led to its widespread use and provides a standard abstraction for computing resources.

However, the question of what abstraction should be provided for the network resources is still being debated. The current approaches differ in the level of detail that they expose to the individual tenants. The "one big switch" approach is the simplest of these abstractions, implemented by systems such as Amazon EC2 [2] with some additional capabilities, and provides the advantage that it requires the least amount of management, as all the virtual machines find themselves connected to a single virtual switch. However, this approach also offers the least capabilities in terms of requesting special functionality that allows the tenant to exploit patters in their traffic.

Nicira [17] uses SDN to allow the tenant to program the switch, allowing more management of individual packet flows, but not of the network configuration at large. However, as research such as Oktopus [8] suggests, information about topology can be used to increase the performance of the virtual networks. As more move to these multi-tenant datacenters, different tenants can have different requests that are not covered by these abstractions. While we can extend the specific abstractions in any number of ways, a more fundamental approach will provide easier management in the future.

Rather than the infrastructure provider offering a particular abstraction, we argue the abstraction should be decoupled from virtualization. Virtualization enables sharing of the infrastructure, whereas the abstraction tells how the tenants use the network. We propose a virtualization layer which provides the ability for tenants to create virtual network and then layer a particular abstraction on top. We build on top of a SDN-based physical infrastructure in order to support both custom SDN application in addition to traditional network equipment (e.g., using SDN-based switches as routers[15]).

In order to support a general virtualization layer, in the subsequent subsections we discuss the requirements for virtualization in terms of (i) specification of the virtual infrastructure, (ii) isolation between virtual infrastructures, and (iii) management of the physical infrastructure that is decoupled from the management of the virtual infrastructure.

2.1 Custom virtual topology and SDN controller

In order to have support the widest variety of tenant requirements, the tenant should be able to design their own custom network as if it was the tenants own physical infrastructure. To do this, the tenant specifies both the topology and custom controller software.

Virtual topology: Each virtual topology consists of nodes, interfaces, and links. The topology as a whole is a collection of virtual nodes which each has a set of virtual interfaces. The virtual interfaces are then connected to other virtual interfaces with virtual links. Importantly, the specification does not include any of virtual addressing as each virtual topology is governed by its tenant's application software independently.

Virtual nodes can be either a server or an SDN-based switch and includes a constraint representing some processing capabilities. In the case of a switch, this constraint is the maximum number of flow table entries to be installed on the switch. In the case of a server, we roughly categorize a physical server's capabilities into units based on number of cores and memory size. A virtual node can then be requested in discrete units. Each virtual link also has a capacity constraint attached to it. This constraint expresses the maximum bandwidth that will be used by this topology on this link and any latency requirements of the link.

Custom SDN controller: In software-defined networking, a logically centralized piece of software known as the controller manages the collection of switches which handle the traffic. This software is provided by each tenant to control the virtual network of switches. The tenant can develop their own controller application and submit it to run as software object on a physical controller. Doing so requires minimal changes to the code of the controller, with the only addition being the remapping of their function calls for interaction with the network to go through the virtualization layer, which add encapsulation to the packet headers as needed.

Should the tenant want a simpler representation of the network, they can choose from default implementations of controllers, such as a simple learning switch application implementing standard Ethernet, or any potentially developed controller that they could use without having to write their own code. As such, the tenant can decide whether they want to have full control of the network, but have to manage individual packet and flow installations with their own software, or deploy an abstraction that is already existing, but give up some of the management capabilities.

2.2 Address space and bandwidth isolation

With networking, the address space is defined by the fields in the packet headers (*e.g.,* source IP address, destination TCP address). This address space is what the SDN controller applications use to define how packets are handled within the network. As such, two applications cannot use the same addresses. To solve this, we *virtualize the address space* by presenting virtual address spaces to each application and provide a mapping between the virtual addresses and the physical addresses. This approach enables each application to have control of all fields within the header (*e.g.,* two applications can use the same IP addresses). This approach is essential in a multi-tenant environment where each tenant controls their own view of the network. To map the virtual addresses into the physical address space transparently to the hosts and controller applications, encapsulation can be used at the edge of the network.

In addition to address space isolation, the virtualization solution must support bandwidth isolation. While current SDN hardware does not include the ability to limit bandwidth usage, the recent specification of OpenFlow includes this capability [7]. Using embedding algorithms, we guarantee bandwidth to each individual virtual link. As support for enforcing these allocations becomes available, we can incorporate them into our system.

2.3 Virtual network decoupled from the physical network

We map the virtual topology presented to the tenant to the physical topology by mapping each virtual node to a unique physical node in the physical network. The mappings are not exposed to the tenants, and are not important to them, as the tenants see their virtual network without directly accessing the physical network. With this decoupling, we can potentially mask things such as link failure or maintenance by remapping the assignments without notifying the tenants of failure. Nodes, both switches and VM's, can move without the tenant's view of the network changing, both in cases of failure and maintenance or load balancing. In addition, enabling arbitrary topologies allows for a single virtual network which can be used in many settings – *e.g.,* a network used in a lab for training how to configure and manage can then be used in the field.

3 The FlowN Architecture

The FlowN architecture is based around two design decisions as discussed in Section 1. In particular, FlowN enables tenants to write arbitrary controller software that has full control over the address space and can target an arbitrary virtual topology.

In this section we present the overall architecture of FlowN, depicted in Figure 1.

3.1 SDN based Physical Network

The FlowN architecture is based around the use of software-defined networking. With these, each physical switch implements a standard API (*e.g.,* OpenFlow [14]) that enables programming of the packet processing capabilities (*e.g.,* a table that determines how to forward traffic). Using this API, arbitrary software can be written to control the switches. This can be performed by a remote

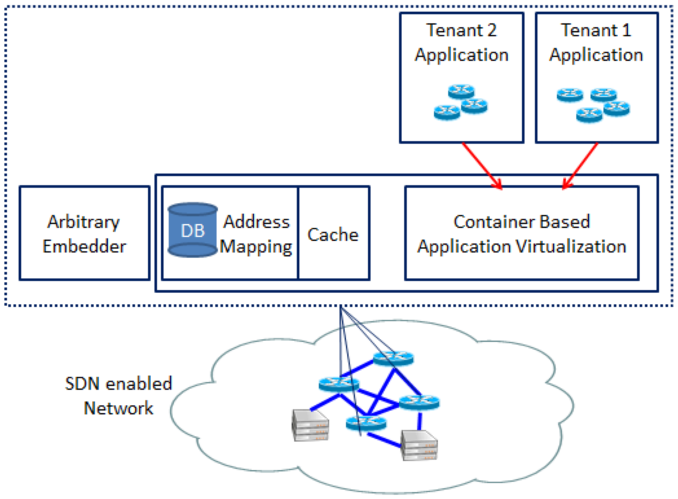

Figure 1: System design

server through a secure communication channel with the switch. This is in contrast to traditional networks in which proprietary software is running on the switches and only a configuration language to configure protocols is provided. With FlowN, the physical network consists of one or more SDN enabled switches each communicating with the logically centralized FlowN based controller. Within the controller, each tenant's controller application is run.

3.2 Tenant Applications

Each tenant provides the FlowN system with the topology that is being requested and at setup time, FlowN will allocate physical resources for that virtual network. Currently, we do not support dynamically modifying the virtual network requests such as changing adding extra nodes and links or increasing virtual links. We intend to address this as future work. The virtual network specification and the controller were discussed in Section 2.2.

3.2.1 Embedding

During the setup of a virtual topology in FlowN, the specification is first passed to the topology parser. The parser will read through the configuration of the virtual topology and build the view of the network without admitting it to the physical network. The embedder then uses an algorithm to map virtual resources to available physical resources.

The choice of embedding algorithm is left up to the datacenter owner, and different embedders can exhibit different properties that might be desirable. Any embedder must obey the following rules: each virtual node of a virtual network is mapped to a unique physical node in the physical network, and each virtual link maps to a path in the physical network, represented by a series of physical links and a hop number. Management of splitting virtual switches across many physical ones or mapping multiple virtual switches to one physical switch is not supported in the virtualization layer, but can be potentially presented as an abstraction to the tenant by embedding a virtual topology and using proper abstractions for the given tenant.

Supporting a software virtual switch co-located with the virtual machines on each server complicates the embedding of the links between virtual hosts and the edge virtual switch in that (i) the bandwidth is dependent on CPU cycles dedicated to the software switch to handle packet processing, not any actual bandwidth, and (ii) a host may be connected to multiple edge switches in the topology. We assume a software switch can be characterized in terms of virtual interfaces which each support a fixed bandwidth – exactly similar to hardware based switches. We also assume each physical server can bypass the software switch, so the link capacity constraint is then based on the physical network interface capacity. To the embedder, there is no difference whether the switch is software or hardware as that will be captures by the constraints associated with the given node.

3.3 Container Based Controller Virtualization Layer

Once the tenant's virtual topology is accepted into the network, the tenant's application can run. These applications are written as if they are targeting a dedicated SDN based network with some particular controller platform (*e.g.,* NOX). The controller platform would maintain the communication channels with the individual SDN enabled switches and provide an API to applications to provide the functionality to deal with different events.

In FlowN, we extend the capabilities of the controller platform to enable multiple applications to run concurrently. That is, the tenant application still is making function calls and is receiving callbacks. However, the underlying controller platform is performing more functions to enable multiple applications to all independently call the same set of functions and receive similar callbacks. For any packet that is received by FlowN from a SDN enabled switch, we perform a lookup to determine the tenant that the packet belongs to. FlowN then calls the appropriate function in the tenant's controller application. If the tenant's application wants to send packets or install flow rules, they call a function that is intercepted by FlowN which translates the virtual address space to the physical address space (as discussed in Section 3.4. FlowN may also do other translations such as changing a request to flood a particular packet on all ports to a series of requests to the physical switches to forward the packet on the individual links that represent the virtual links (since a physical switch might have some ports used by one customer but not others).

This approach to using a direct function call and callback API is in contrast to using the protocol used to communicate with the individual switches as the mechanism. For example, we could have used the OpenFlow communication protocol between FlowN and individual tenant applications and let each tenant run their own OpenFlow controller. The distinction is similar to the different of container based virtualization and hypervisor based full virtualization in server virtualization. With container-based virtualization, the kernel support running multiple, isolated user-space environments (or containers). The kernel is shared but provides the containers with independent namespaces and resource scheduling. With full virtualization, the virtual machines each run their own operating system and the virtualization layer makes it appear as though each virtual machine is running on dedicated hardware. FlowN is analogous to container-based virtualization since as they are not running their own instances of the controller platform. FlowVisor [18], for example, is analogous to full virtualization as the application do run their own controller and speak the OpenFlow protocol with the FlowVisor virtualization layer. As controllers become more complex and more layered, the efficiency of a container based solution becomes more important.

3.4 Virtual-Physical Address Space Mapping

To provide each tenant application with its own address space, the FlowN software must provide a mapping between virtual and physical resources. We discuss our use of a database to determine the mapping in Section 4 and discuss the virtual address mechanism here.

Similar to Nicira, we use encapsulation to enable each tenant to have its own address space. In particular, we provide tunning for each physical link. While we could simply encapsulate each packet at the edge of the network with a header that is unique to each tenant, the number of tenants supported will then be limited by the number of bits in this header. Instead, with FlowN each physical link has its own virtual address space. That is, we encapsulate each packet as it enters the network, and perform label swapping within that header at each switch in the network. With this, packets received at a switch can be classified based on the (i) physical interface, (ii) label in the encapsulation header, and (iii) fields as specified by the tenant application.

A virtual to physical mapping occurs when an application makes a modification to the flow table (*e.g.,* adding a new flow rule). We append to each rule information to make it so the rule is unique. A physical to virtual mapping occurs when the physical switch sends a message to the controller (*e.g.,* when a packet does not match any flow table rule). In the case when multiple tenants are affected by a single event (*e.g.,* link failure), multiple physical to virtual mappings will occur.

4 Database for Virtual to Physical Mapping

To store the current state of the virtual to physical mappings, we use a database, which consists of three parts - the virtual topology descriptions, the current available physical topology resources (unused link capacities and free virtual machine slots), and the mappings of virtual topologies to the physical network.

The physical topology is represented in the same fashion as the virtual topologies as described above, except for the link capacities representing the remaining bandwidth allocations, switch capacities representing the remaining switch rule capacities, and the virtual machine slots on the physical servers having the property of being either currently used or not.

Both the topology descriptions and the assignment lends itself directly to the relational model of a database. Each virtual topology is uniquely identified by some key, and consists of a number of nodes, interfaces and links. Nodes house the corresponding interfaces, and links connect one interface to another. The physical topology is described in a similar fashion. Each virtual node is mapped to one physical node; each virtual link becomes a path, which is a collection of physical links and a hop counter giving the ordering of physical links.

4.1 Simplifies Mappings

Mappings are stored as two tables within the database. The first table stores the node assignments, mapping each virtual node to one physical node on which it resides. The second table stores the path assignment, by mapping each virtual link to a set of physical links, each with a hop count number that increases in the direction of the path.

The implemented OpenFlow actions and the corresponding actions on the database are shown in Table 1. The queries given in this table are simplified to use the NATURAL JOIN notation, which will join separate tables based on fields with common names. In practice, this approach is often considered potentially harmful as it obscures which fields the joins happen on - explicit equalities are specified in the prototype.

Most of the remaining OpenFlow actions are realizable with similar queries. As an example, in the case of an aggregate flow stats request, which gives statistics about all flows matching some set of fields on a given output port, we can remap the virtual switch and port id's to the physical switch and set of ports, and request multiple flow statistics requests, one matching each possible encapsulation tag to be used, as well as the virtual network fields. Afterwards, we collect the responses, remove the encapsulation tags, and report them to the virtual network.

Some actions, however, are unrealizable in this approach. OpenFlow 1.0 features a queue statistics request, which provides queue statistics for one or more ports and queues on a switch, is not realizable as there is no way of uniquely distinguishing the output queues between different virtual networks.

OpenFlow action/event	Description	Database Lookup

Packet received events(from VM)	If packet came from a VM, translate switch id and the port id it arrived on to the corresponding virtual tenant, switch and port, since we discretize VM assignment.	SELECT L.customer_ID, L.node_ID1, L.node_port1
		FROM Customer_Link AS L NATURAL JOIN (Path_P2C_Mapping AS M, Physical_Link AS P), Customer_Node AS C
		WHERE P.node_ID1 = x AND P.node_port1 = y AND C.node_type = ”server” AND C.node_ID = M.node_ID1

Packet received events(internal on network)	If the packet is internal to the network, translate the VLAN tag, switch id and port to the corresponding virtual tenant, switch and port	SELECT L.Customer_ID, L.node_ID1, L.node_port1
		FROM Customer_Link L, Node_C2P_Mapping M
		WHERE VLAN_tag = x AND M.physical_node_ID = y AND M.customer_ID = L.customer_ID AND L.node_ID1 = M.customer_node_ID

Send single packet, or install/remove single output flow	Translate virtual switch and output port to physical switch and output port, and the VLAN tag to encapsulate with	SELECT L.VLAN_tag P.node_ID1, P.node_port1
		FROM Customer_Link AS L NATURAL JOIN (Path_C2P_Mapping AS M, Physical_Link AS P)
		WHERE L.node_ID1 = x AND M.customer_ID = y AND L.node_port1 = z

Send broadcast packet, or install/remove broadcast output flow	Translate flood on a virtual switch to a physical switch series of physical output ports. Exclude input port (as floods do).	SELECT L.VLAN_tag P.node_ID1, P.node_port1
		FROM Customer_Link AS L NATURAL JOIN (Path_C2P_Mapping AS M, Physical_Link AS P)
		WHERE L.node_ID1 = x AND M.customer_ID = y AND NOT L.node_port1 = z

Switch up/down events	Translate physical switch to virtual switches for all tenants using this switch	SELECT customer_ID, customer_node_ID
		FROM Node_C2P_Mapping
		WHERE physical_node_ID = x

Port up/down events	Translate physical switch and port to a list of all virtual switches and ports using this port.	SELECT C.customer_ID, C.node_ID1, C.node_port1
		FROM Customer_Link AS L NATURAL JOIN (Path_C2P_Mapping AS M, Physical_Link AS P)
		P.physical_node_ID1 = x AND P.physical_node_port1 = y

Table 1: OpenFlow events or actions available to the virtual networks and the corresponding simplified database lookups that occurs to fulfill them.

4.2 Leverage Database Research

Besides the simplicity of using the relational model to represent network topologies and mappings, we leverage existing database research to gain consistency and durability properties about the network state they represent. Each transaction brings the database from one valid state to another, so partial state from new network embedders or potential reassignments will not be seen until the full transaction has complete. Database state is also guaranteed to be persistent bar complex disk failure. If the same state was stored in the controller or a program running in memory, the state of the network would depend on one or multiple machines uptime, which could potentially be lost in a crash or network split.

Another advantage is the behavior of the data. Since our database stores the mapping of a virtual network to a physical network, we expect many more reads than writes in this database. In this case, we can set up a master database server that would get any of the updates, which are done in case of topology changes, admitting new virtual networks or network element failure, none of which are expected to occur frequently during normal operation. Multiple slave servers can be used to replicate the state. This would help more so if the size of the physical network demands multiple virtualization controllers for the OpenFlow network, as each could maintain it's own database copy locally. Since mappings are done on a per-switch basis, each physical OpenFlow switch can thus talk to it's own OpenFlow controller and the database will handle consistent network state across all such controllers.

5 Evaluation

Each of the mappings described in Section 4 adds some overhead to the processing on the controller. In this section, we evaluate this overhead.

5.1 Prototype and Setup

We built a prototype of FlowN by extending the NOX version 1.0 OpenFlow controller [12]. This controller runs without any applications initially, instead providing an interface to add applications (*i.e.,* for each tenant) at run time. This includes performing the embedding step, which we base our algorithm from Chowdhury, et. al. [10]. The embedder populates a MySQL version 14.14 based database with the virtual to physical mappings. We implement all schemes using the InnoDB engine. For encapsulation, we use the VLAN header, pushing a new header at the ingress of the physical network, swapping labels as necessary in the core, and popping the header at the egress. Alongside this controller, we also run a memcached [3] instance which gets populated with the database lookup results and provides faster access times should a previous lookup be repeated.

We run our prototype on a virtual machine running Ubuntu 10.04 LTS given full resources of 3 processors of a i5-2500 CPU @ 3.30GHz, 2 GB of memory, and situated on a SSD drive (Crucial m4 SSD 64GB). We perform tests by simulating OpenFlow network operation on another VM (running on an isolated processor with its own memory space) through either mininet [13] or cbench [6] modified to properly generate packets with the right encapsulation tags.

We performed two sets of experiments, one in latency for packets arriving at the controller, and one for failure outage times when a link fails in the topology.

5.2 Latency

The latency experiment measures scalability in the number of virtual networks for packet processing on one physical controller.

For this experiment, we modified cbench [6] to simulate packet-in events in the physical network with the appropriate VLAN tags if they are on the inside of it. The packets otherwise contain a unique MAC address. The virtual network controllers for each network are simple learning switches that operate on individual switches. Due to the MACs being unique, each new packet-in event will trigger a rule installation. For the latency test, the cbench application waits until the rule installation has been received before sending out the next packet-in event to the controller. We compare our system against unvirtualized NOX, where a single learning switch is running the entire network, and FlowVisor, which runs multiple virtual network controllers, each receiving an even fraction of traffic based on VLANs tags.

We performed random pre-caching, whether happening in memcached or in the InnoDB buffer pool, by running several trials before recording results from multiple experiments and recording the number of flow installations that occurred each second. We measure the latency as the inverse of that number.ding the number of flow installations that occurred each second. We measure the latency as the inverse of that number.

The results are presented in Figure 2. We notice that while our application has mostly uniform latency based on the number of virtual networks, FlowVisor exhibits linear growth of latency in the number of virtual networks. For the largest case of 100 virtual networks, our prototype performs better than FlowVisor. Extrapolating on these results, for large amounts of virtual networks (as is the case in datacenters), we expect to perform better than FlowVisor. We also note that the overall increase in latency over the unvirtualized case is less than .2ms for the prototype using memcached and .3ms for the one without.

Since our application is currently running on a single thread, the results for the throughput test, which does not wait for the rule installation to happen for a packet, but rather queues packets up until the buffer is full, is very similar.

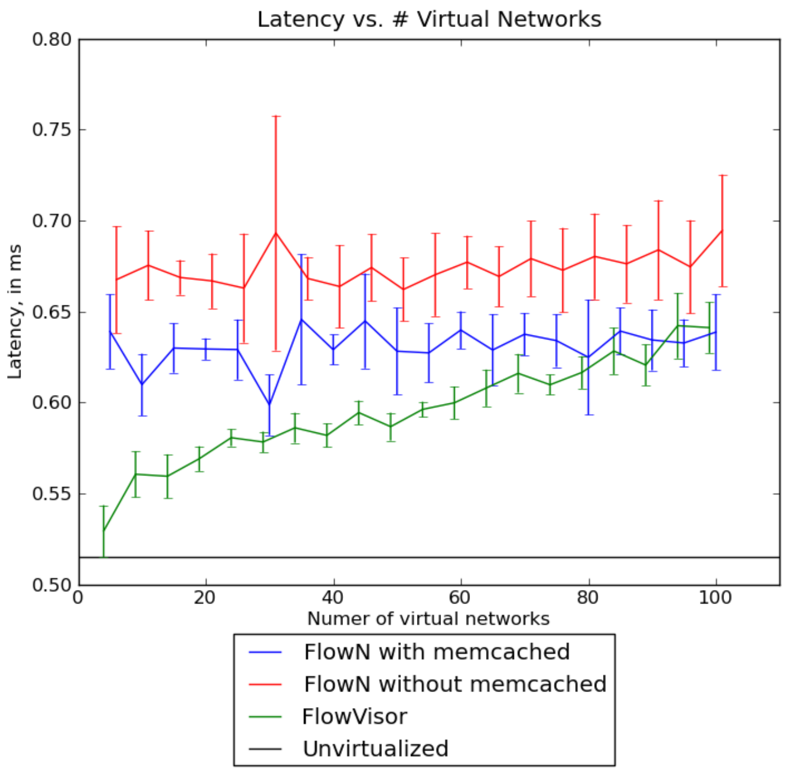

Figure 2: Latency vs. Virtual Network count

5.3 Failure Times

The failure experiment measures scalability in the number of virtual networks in the event of a link failure where multiple tenants use the link for communication.

For this experiment, we run mininet [13], a OpenFlow network simulation testbed, to simulate 5 physical switches and 5 hosts. We dedicate one thread to running a high-speed ping every 1 ms between two hosts on this network. We then bring down a link on this network that is on the shortest path between the two hosts running pings. The hosts belong to each of the virtual networks, and each virtual network controller is aware of the location of the hosts and performs shortest-path computations to route traffic between them. When a link goes down, this triggers updates in all the virtual network controllers to install the new shortest path between each pair of hosts.

We measured the timeout period for the ping between the hosts as a measure of average fault time for the virtual networks.

The results are presented in Figure 3. If the virtual network the hosts belong to is one of the first to install its rules, the failure time can be small even for large amounts of virtual networks. The failure outage times for 50 virtual networks never exceeded 65ms.

6 Future Work

As this work has been exploring a relatively novel field, there are many possible points of expansion for this project. Here, we address some of the important ones for the development of a functional system that fully meets our requirements.

6.1 Modifying existing virtual networks

Our current prototype is able to admit and remove virtual networks into the system, but not modify them incrementally. For datacenter environments, the ability to quickly adjust virtual networks by adding extra hosts and network elements is crucial as ease of adaptation to processing power demand shifts is an important factor in transitioning to a datacenter. Being able to do so without readmitting the entire virtual network efficiently is an important factor for future work.

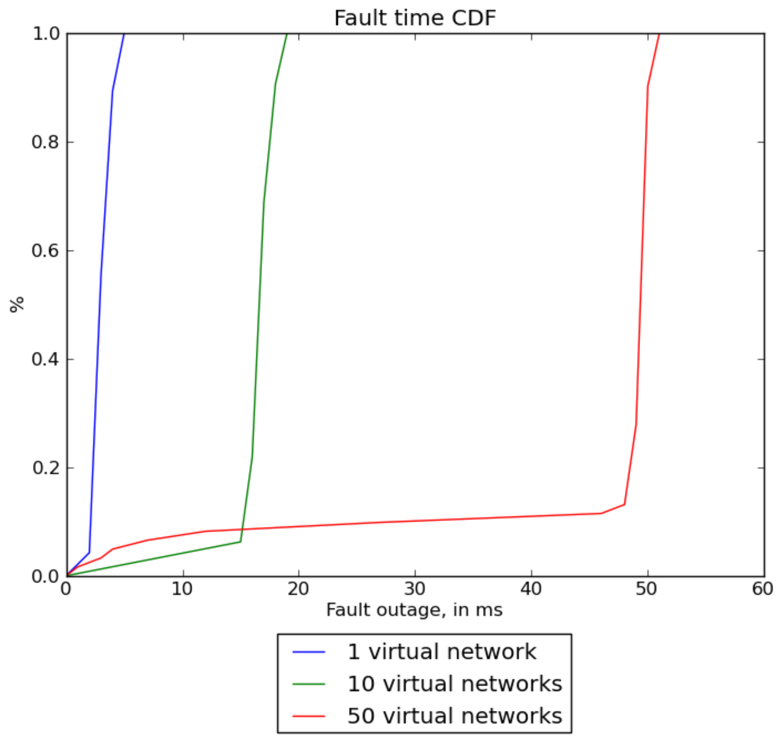

Figure 3: Fault times vs. Virtual Network count

6.2 Replication of virtualization layer

Our current prototype is running a single instance of the virtualization layer on one physical controller. To fully comply with the requirement of scalability in the size of the physical network, we (and any SDN project) must allow multiple physical controllers to manage separate parts of the network. We propose to run different physical controllers for different parts of the physical network, each controlling some number of switches and housing some amount of virtual networks. The virtual networks housed on a given controller do not need to be confined to only have virtual nodes mapped to the subset of the physical topology managed by the corresponding physical controller, instead, controllers can communicate amongst each other.

Should a controller receive a packet which is destined for a virtual network not housed on the controller, it can determine which physical controller to forward it to through a mapping of virtual network identifiers to physical controllers. It can then forward the message to the appropriate physical controller, where the packet will undergo the physical to virtual translation, get processed by the virtual network, and potentially invoke packet sends or flow installations. The physical controllers can communicate in a similar fashion to take the appropriate actions on the right subset of the network.

6.3 Replication of database

Our current prototype is using a single MySQL database. With multiple physical controllers, even in the presence of caches adjacent to each of them there exists a possibility, especially as new networks are admitted or old networks are modified, that a period of time exists where each of them needs to communicate to the database. Databases already support efficient replication algorithms, and using one of them to have multiple instances will improve latency for peak load times on the database.

6.4 Potential embedding methods and guarantees

We presented one potential implementation of the embedder, based on work from Chowdhury, et. al. [10]. This embedder performs a LP relaxation of a mixed integer problem, where the goal is to figure out the optimal assignments that minimizes an objective function which assigns weights to each node assignment or link assignment based on the capacities consumed compared to the remaining capacities on the links and nodes. Such an assignment will prefer links and nodes with high remaining capacity over low, and otherwise attempt to place adjacent virtual nodes as close as possible, and have virtual links traverse short paths in the physical network.

However, if the tenant has certain constraints on the latency of network operation or the proximity of certain VM's, other potential assignment algorithms can be used, which attempt to find mappings that obey the tenant constraints. A potential problem is to devise a set of different algorithms, each with different guarantees that they can provide, which can be used by the datacenter owner to perform the embedding based on some constraints that the tenant can specify.

7 Related Work

Virtualization as the subdivision of some single physical resource into virtual parts that are used by different entities has seen many applications in computer science, including networking. Virtualization if individual components is a well developed area, with perhaps the most basic application of being the development of virtual machines. Virtual machines allow partitioning of physical resources amongst many concurrently operating, isolated machines, which can be beneficial for common use in cases where individuals want to run separate systems, or in datacenters, where VMs are leased out to tenants. Many open source solutions (*e.g.,* VirtualBox [4]) or commercial products (*e.g.,* VMWare [5]) exist in this area. Virtualization of routers is supported by major vendors (*e.g.,* Cisco [1]) and provide the ability to isolate different routing entities, easing management and reducing resource consumption - such developments are typically targeting the data center environment.

Virtualization of groups of network elements has been researched quite widely as well. Virtual Private Networks provide isolation between a set of machines across a wide area network, but restricted to point-to-point link tunnels, and Virtual LANs provide the ability to isolate different types of traffic in a local area network, but restricted to having any machine be only identified by one VLAN, as well as utilizing a subset of the existing topology for each VLAN. More general approaches to network virtualization have to support arbitrary virtual topologies.

7.1 Non-SDN virtualization

Full network virtualization, or topology switching, has been researched in the context of non-SDN environments. Depending on the nature of the environment, many different projects have been proposed. For example, for the research testbed, the VINI project [9] can be used to build a virtual network infrastructure spanning a wide area. Each virtual network in VINI can be used as a research network to test new protocols or services on an arbitrary network by being able to send traffic and manage packet forwarding using routing software on nodes designated as network elements. In the context of ISP-scale network management, the Cabo [11] project aims to separate the infrastructure provider from the service provider by introducing virtual networks as an architecture that spans many different providers. Closest to our work, in the datacenter environment, research such as the Topology Switching [19] paper has been exploring how to efficiently allow individual applications to specify their needs for a particular topology, including isolation, bandwidth and resilience constraints.

7.2 SDN virtualization

While a lot of work has been done for network virtualization in the SDN environment, the current solutions differ from our approach in their approach to splitting the address space and virtual topology representation.

There are two current approaches to addres space. First, a virtualization layer can *partition the address space* and assign arbitrary slices of this “flow space” to individual applications – an approach taken with FlowVisor [18] and PFlow [16]. Second, a virtualization layer can *virtualize the address space*. This is the approach taken with Nicira [17] which enables each application to have control of all fields within the header (*e.g.,* two applications can use the same IP addresses). To map the virtual addresses into the physical address space transparently to the hosts and controller applications, Nicira utilizes the ability to perform encapsulation at the edge of the network. While each approach certainly has use in different scenarios, for a multi-tenant environment where each tenant controls their own view of the network, virtualizing the address space is the more appealing solution as address space choices between tenants do no conflict with each other.

The virtual topology can be presented to individual tenants in different ways. One approach, as taken with Nicira [17], is to abstract away the internals of the network and present a topology consisting of *edge switches with full connectivity mesh*. This approach enables applications to provide functionality that can be performed at the edge, such as access control and quality of service, without having to deal with how packets are actually handled (the virtualization layer manages the network between each edge switch to present the abstraction of a series of point to point links). A second approach is to present applications with a *sub-set of physical topology*. This approach, as taken by FlowVisor [18] and PFlow [16], enables the applications to have more control of how traffic is handled. For example, a public cloud infrastructure can allocate guaranteed bandwidth to each virtual network and allow the individual tenants to decide how they allocate that bandwidth. This is in contrast to the edge switches with full connectivity mesh topology in which the public cloud provide would control how all traffic is handled. Our approach of a *arbitrary virtual topology* provided to each application, however, is novel.

8 Conclusions

In this paper we presented FlowN which provides a full network virtualization solution for software defined networks. The FlowN architecture is based around the use of namespaces where the controller platform is shared by each virtual network for an efficient solution. We make use of database technology which has been developed to support such features as replication and caching, so that we do not need to implement those in FlowN. This enables the mappings between virtual and physical in a efficient and scalable manner. We presented, and evaluated, a prototype that performs the virtualization, and discussed the future work that would enhance it.

References

[1] Router virtualization in service providers, 2008. See `http://www.cisco.com/en/US/solutions/collateral/ns341/ns524/ns562/ns573/white_paper_c11-512753.pdf`.

[2] Amazon EC2. See `http://aws.amazon.com/ec2/`, November 2010.

[3] Memcached: A distributed memory object caching system. See `http://www.memcached.org`, November 2010.

[4] VirtualBox. See `https://www.virtualbox.org/`, November 2010.

[5] VMWare. See `http://www.vmware.com/`, November 2010.

[6] Openflow operations per second controller benchmark. See `http://www.openflow.org/wk/index.php/Oflops`, March 2011.

[7] Openflow switch specification version 1.3.0 (wire protocol 0x04), June 2011. See `https://www.opennetworking.org/images/stories/downloads/openflow/openflow-spec-v1.3.0.pdf`.

[8] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter

networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 242–253, New York, NY, USA, 2011. ACM.

[9] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: realistic and controlled network experimentation. *SIGCOMM Comput. Commun. Rev.*, 36(4):3–14, August 2006.

[10] N. Chowdhury, M. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783 –791, april 2009.

[11] N. Feamster, L. Gao, and J. Rexford. How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, January 2007.

[12] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an operating system for networks. *ACM SIGCOMM Computer Communications Review*, 38(3), 2008.

[13] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *ACM SIGCOMM HotNets Workshop*, pages 1–6, 2010.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communications Review*, 38(2):69–74, 2008.

[15] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corra, S. C. de Lucena, and M. F. Magalhes. Virtual routers as a service: The RouteFlow approach leveraging software-defined networks. In *International Conference on Future Internet Technologies*, June 2011.

[16] NEC. ProgrammableFlow Controller. `http://www.necam.com/PFlow/doc.cfm?t=PFlowController`.

[17] Nicira. Network virtualization platform. `http://nicira.com/en/network-virtualization-platform`.

[18] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Operating Systems Design and Implementation*, October 2010.

[19] K. C. Webb, A. C. Snoeren, and K. Yocum. Topology switching for data center networks. In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE'11, pages 14–14, Berkeley, CA, USA, 2011. USENIX Association.