A Recursive Approach to Network Management

Yuefeng Wang Ibrahim Matta
Computer Science Department, Boston University
{wyf, matta}@bu.edu

***Abstract*—Nowadays there is an increasing need for a general management paradigm which can simplify network management and further enable network innovations. In this paper, in response to limitations of current Software Defined Networking (SDN) management solutions, we propose a recursive approach to enterprise network management, where network management is done through managing various Virtual Transport Networks (VTNs). Different from the traditional virtual network model which mainly focuses on routing/tunneling, our VTN provides communication service with explicit Quality-of-Service (QoS) support for applications via transport flows, and it involves all mechanisms ($e.g.$, routing, addressing, error and flow control, resource allocation) needed to support such transport flows. Based on this approach, we design and implement a management layer, which recurses the same VTN-based management mechanism for enterprise network management. Comparing with an SDN-based management approach, our experimental results show that our management layer achieves better network performance.**

I. Introduction

Traditionally network management is a complicated and error-prone process that involves low-level and vendor-specific configurations of physical devices. Nowadays computer networks have become increasingly complex and difficult to manage, and new cloud-based service models [1] have become the norm in networking economics. These trends increase the need for a general management paradigm to simplify network management and further enable network innovations. Making the network programmable is an efficient way to address the complexity of network management by providing high-level network abstractions and hiding low-level details of physical devices. Software Defined Networking (SDN) [2] has drawn considerable attention due to the popularity of OpenFlow [3]. SDN focuses on programming the control plane through a network management layer and has been widely deployed in enterprise and data center networks [3], [4]. However SDN management layers are plagued with limitations inherited from the TCP/IP architecture [5].

In response to these limitations, we propose an application-driven recursive approach to enterprise network management. In our approach, network management is done through managing various Virtual Transport Networks (VTNs), inspired by and built atop a new network architecture, RINA [6], [7], which aims to solve current TCP/IP limitations[1]. Different from the traditional virtual network model which mainly focuses on routing/tunneling, a VTN provides communication service with explicit QoS support for applications via transport flows, and it includes all mechanisms ($e.g.$, routing, addressing, error and flow control, resource allocation) needed to support such transport flows. Furthermore VTN is application-driven, where a VTN can be dynamically formed and instantiated with different policies to meet different application-specific requirements. One of the biggest advantages of VTN is that it allows both flow aggregation and splitting, where we can either aggregate multiple flows into one single flow or split one flow into multiple flows, and it allows better resource allocation and utilization in support of various requirements.

The contributions of this paper are: (1) we propose an application-driven recursive approach to enterprise network management, where network management is done through managing various Virtual Transport Networks (VTNs); (2) we present the design and implementation of a management layer[2] based on our approach, which allows managing and dynamic formation of such VTNs to support flow requests from regular users and to meet application-specific requirements; (3) we propose the VTN formation problem which aims to improve network performance, and then give a practical solution to it; and (4) we show the advantages of VTN-based management through experimental results. The rest of the paper is organized as follows. We review related work of SDN-based management solutions in Section II. Details of our VTN-based management approach are explained in Section III. Network API and details of the protocol supporting our management layer as well as implementation of our management layer are described in Section IV and Section V, respectively. The VTN formation problem and its solution are presented in Section VI. Performance evaluation is presented in Section VII. In the end, we conclude this paper with future work in Section VIII.

II. Related Work

The core of an SDN-based management solution is the management layer (such as [8], [9], [10], [11], [12]), and itself does not manage the network but provides a global network view and general programming interface (the so-called "Northbound" API [13]) to management applications (designed or programmed by network managers), which actually manage the network. Access control, virtual machine (VM) migration, traffic engineering, and routing are examples of management applications. The network management layer translates high-level network policies specified by the management applications into low-level and vendor-specific configurations of network devices (switches or routers) through the OpenFlow protocol [3] (or any of so-called "Southbound" API [13]).

SDN reduces management complexity by providing high-level network abstractions. However most SDN management

[1] A VTN is termed a DIF in [6], [7], to mean a Distributed Inter-Process Communication (IPC) Facility.

[2] In this paper, we use the terms "management platform", "management layer" and "control platform" interchangeably.

layers (such as [8], [9]) only provide the management-level interface, used by network managers to write management applications to monitor and control the network. And they lack a user-level interface, used by regular users to write general applications (such as video application) and achieve better performance for their applications. Another limitation with SDN management layers is their lack of Quality-of-Service (QoS) support as a consequence of their reliance on the current TCP/IP architecture which is designed to only provide best-effort service. QoS support is important because it can not only improve the performance of user applications via guaranteed services but also improve network performance via better resource allocation. Although some management layers (such as [12]) attempt to provide QoS support, so far there are no common SDN mechanisms for QoS support.

Network virtualization allows multiple isolated virtual networks to be built on top of the same physical infrastructure. It can improve resource utilization through network consolidation and provide isolation for security purposes or for developing and testing new network features. Some SDN management layers (such as [10], [14], [15], [16]) support network virtualization, but they mainly focus on routing and access control, and do not consider other mechanisms (such as error and flow control and resource allocation) for transport purpose, which is important for network resource utilization.

Essentially SDN is still a flat management solution which lacks levels of management scope, and every component (physical components such as routers or switches, and logical components, $i.e.$, processes) is part of the same and only management scope. It is not easy to dynamically define new (higher-level) management scopes over existing management scopes. Most of these problems are due to reliance on the TCP/IP architecture [5], which notably lacks resource allocation and flow/error control over limited scopes. There is existing SDN work that provides recursive control ($e.g.$, [17], [18]), but it lacks transport-level flow/QoS control over limited scopes and does not support dynamic management of these scopes.

III. Design

In this section, we explain the design of our VTN-based approach for enterprise network management, and present the components of our management layer. Different from SDN which is mainly based on the TCP/IP architecture, our approach is inspired by and built on top of a new network architecture, the Recursive InterNetwork Architecture (RINA) [6], [7]. RINA is based on the principle that networking is Inter-Process Communication (IPC) and only IPC. RINA solves shortcomings of the TCP/IP architecture by addressing the communication problem in a more fundamental and structured way. RINA provides communication services via transport flows by using a recursive building block (which we call VTN). The building block involves all kinds of mechanisms ($e.g.$, enrollment, authentication, routing, addressing, error and flow control, resource allocation) to support transport flows of a certain scope.

Our contribution over existing work on RINA is that we exploit the usage and benefits of such building block for the purpose of network management, and further optimize its placement to improve network management.

A. Virtual Transport Network (VTN)

A Virtual Transport Network (VTN) is the basic building block in our network management. The job of a VTN is to provide communication service with QoS support via transport flows for user applications. Unlike a regular virtual network which mainly focuses on routing/tunneling, a VTN involves all kinds of mechanisms ($e.g.$, enrollment, authentication, routing, addressing, error and flow control, resource allocation) needed to support transport flows over a certain management scope. A transport flow provides end-to-end communication service with QoS parameters, which differs from a tunnel which is usually hard-coded, and just provides best-effort service over an overlayed routing path (tunnel) without resource allocation, flow and error control.

A transport process is a process that is capable of establishing transport flows across the VTN at requested QoS levels. Each VTN consists of a set of transport processes which run on different hosts (or switches), and the operations of its member processes are contained in the VTN itself. VTN is a secure container, where every process has to be explicitly enrolled through an authentication procedure [6], [7]. Each transport process contains a data transfer component supporting transport flows between different applications. And the VTN provides communication service to application processes by exposing a flow allocation interface.

Each VTN has its management scope, $i.e.$, each VTN includes a limited number of transport processes running on a limited number of physical hosts. And each VTN maintains the mapping between applications and transport processes, $i.e.$, application name resolution within its scope. The same VTN mechanism can be repeated to provide a larger-scope transport service for applications by recursively using the smaller-scope transport service provided by existing VTNs. Namely, we can build VTNs of different levels to provide transport services over different scopes. Different VTNs use the same mechanisms but may use different network policies ($e.g.$, policies for routing and error and flow control), and the transport processes inside the same VTN follow the same policies specific to the particular VTN.

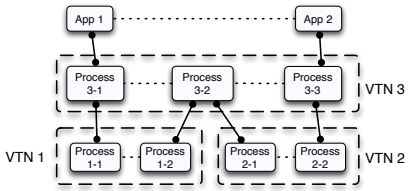


Fig. 1: Two levels of VTNs, and *VTN 3* spans a larger scope. Each process inside a VTN is a transport process.

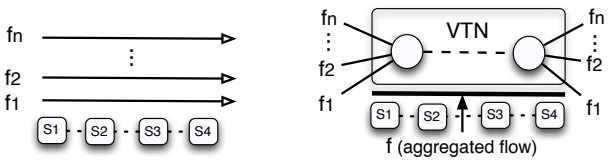
Figure 1 shows a simple example of VTNs providing transport service over different scopes. *VTN 1* and *VTN 2* each spans a smaller scope, and can provide transport services

to applications inside its scope. If an application *App 1* in one scope wishes to communicate with another application *App 2* in another scope, and since *VTN 1* and *VTN 2* cannot satisfy such request, we need a higher level *VTN 3* which spans both scopes and provides a transport service across the larger scope. Recursively, we can repeat VTN to provide an even larger-scope transport service, $i.e.$, any two application processes can communicate as long as a common underlying VTN can be found or built.

One advantage of VTN is that it explicitly provides QoS support when applications request a transport service. It is important to regular users, as they can know what kinds of service they can get ahead of time, and use this information to improve their application performance. It is also important to network managers, as they can predict more accurately resource consumption, do better resource allocation and ultimately achieve better network utilization.

Another advantage is that VTN provides the opportunity to control management complexity. Each VTN has its own scope, and can be managed without interfering with the operations of other VTNs. By allowing the dynamic formation of VTN, we can either break a larger (lower-level) management scope into smaller ones or aggregate smaller scopes into a larger one. This gives us more flexibility for network programmability compared to existing SDN platforms.

What's more, VTN allows flow aggregation which can help reduce the memory usage in switches, as well as achieve better resource utilization. Most OpenFlow switches use TCAM (Ternary Content Addressable Memory [19]) to store flow forwarding entries (rules) to increase packet processing speed, but TCAM is expensive and has limited storage capacity. Consequently, SDN management layers have to deal with the TCAM problem by reducing the number of flow rules. Most SDN work (such as [20], [21], [22], [23]) focuses on flow rules for access control or firewalling, however, due to SDN's reliance on the TCP/IP architecture, nothing much can be done to reduce the number of flow rules for end-to-end routing purpose other than using shortest path routing [24].



(a) (b)

Fig. 2: (a) n application flows going through 4 switches (S1, S2, S3 and S4). (b) n non-aggregatable application flows can be aggregated into one flow by VTN.

A simple scenario is shown in Figure 2(a). For SDN, if these n flows are not aggregatable due to distinct IP prefixes and port numbers ($i.e.$, wildcard rules cannot apply), we need a total number of $4 \times n$ forwarding rules in switches. It gets worse as the number of non-aggregatable flows increases. However with VTN, we can aggregate these n flows into one flow `f` as shown in Figure 2(b), so we do not need a flow rule for each of the n flows in each switch but rather only one rule for the aggregate flow. Thus the total number of flow rules needed are 4 rules for the aggregated flow, and n rules for multiplexing and n rules for demultiplexing, at the source and destination switches, respectively, for a total of $2n + 4$.

Flow aggregation enabled by VTN also improves resource utilization. Consider the example in Figure 2(a) again, where each flow asks for a guaranteed throughput, and let X_i $(i = 1, ..., n)$ be the instantaneous traffic demand of each flow. Assume the instantaneous traffic demand for each flow follows the same uniform distribution, where the maximum instantaneous throughput is max, mean throughput is μ and standard deviation is σ. Assume for each flow we reserve a bandwidth of $\eta \times max$, where $\eta \in [0, 1]$ denotes the effective per-flow bandwidth requirement. Assume the QoS requirement is defined as the probability (denoted by $1 - \epsilon$) that the instantaneous traffic demand for all flows does not exceed the reserved total bandwidth.

For SDN-based management layers without flow aggregation, to satisfy this QoS requirement, we need

$$\prod_{i}^{n} Prob(X_i \leq \eta \times max) > 1 - \epsilon \tag{1}$$

For our VTN-based management layer with flow aggregation, according to the *Central Limit Theorem*, the aggregated instantaneous flow rate follows a normal distribution, and to satisfy the same QoS requirement, we only need

$$Prob\left(\frac{\sum_{i}^{n} X_i}{n} \leq \eta \times max\right) > 1 - \epsilon \tag{2}$$

We can easily see that for the same $1 - \epsilon$, we need a bigger η to satisfy (1) than (2). That means using our VTN-based management layer with flow aggregation, we can satisfy the same QoS requirement with less per-flow bandwidth reservation. Namely we can serve more flow requests given limited link capacity.

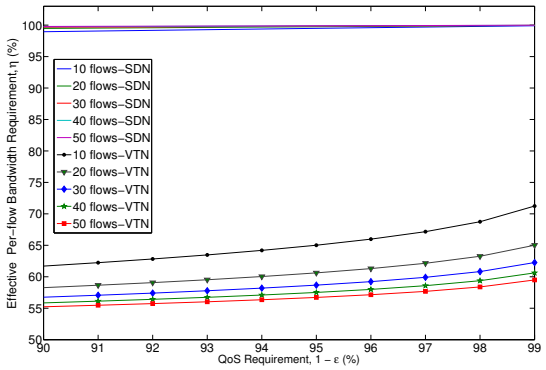


Fig. 3: Effective per-flow bandwidth requirements for different QoS.

Next we show this advantage through an example. Assume on average there are n flows between a pair of switches, which can be aggregated into one flow by a higher-level VTN, and the instantaneous traffic demand of each flow follows a uniform distribution between 0 Mbps and 1 Mbps. Figure 3 shows the effective per-flow bandwidth requirement ($i.e.$, η) to

satisfy different QoS requirements ($i.e., 1-\epsilon$). For the same n, SDN solutions require more effective per-flow bandwidth requirement (almost close to 100% of the peak demand for higher QoS) than our solution. Also Figure 3 shows that with our solution, as the number of aggregated flows increases, the effective per-flow bandwidth requirement decreases to satisfy the same level of QoS[3]. This shows that the more flows that are aggregated, the better performance our management layer achieves. Similar advantage of flow aggregation was also shown in previous work [25].

B. Management Components

Next we explain the components ($i.e.$, distributed applications) for managing (1) a single VTN; and (2) all VTNs.

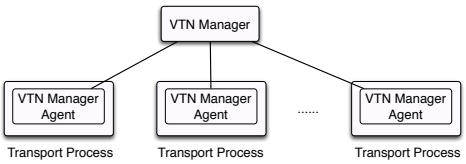


Fig. 4: VTN Manager and its agents for a single VTN.

1) VTN Manager and VTN Manager Agent:

As shown in Figure 4, the distributed application for managing a single VTN includes a *VTN Manager* and its *VTN Manager Agent*. Every VTN has a VTN manager, which is a process that can be implemented in a centralized or distributed fashion, and it manages the whole VTN by specifying different network policies inside the VTN such as routing, access control, and transport policies. A VTN manager agent is part of each transport process inside the VTN, and it exposes a programmable interface for the VTN manager to translate high-level network policies to transport process's configurations.

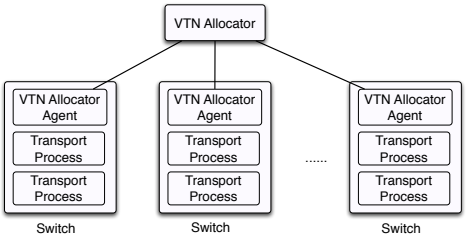


Fig. 5: VTN Allocator and its agents for an enterprise network.

2) VTN Allocator and VTN Allocator Agent:

As shown in Figure 5, the distributed application for managing all VTNs inside the enterprise network includes a *VTN Allocator (VA)* and its *VTN Allocator Agent (VAA)*. The network has one VA, which is a management process that can be implemented in a centralized or distributed fashion, and it manages all VTNs as a whole. Each host (switch) inside the network has a VAA, which exposes a programming interface allowing the VA to create new transport processes on the host and thus build new VTNs across multiple hosts within the network. VA manages the network by managing existing VTNs and building new VTNs dynamically to support different application flow requests.

[3]Proof is skipped due to space limitation, but it is straightforward to prove using the Central Limit Theorem.

3) VTN Resource Manager (VRM):

Every application process has a component called *VTN Resource Manager (VRM)*, which manages the use of all VTNs available to this particular process. It is the job of VAA of the host to decide which VTNs are accessible to particular processes. An application process uses its VRM to allocate transport flows with QoS requirements to other processes, and the VRM in turn passes the flow allocation requests to VTNs via the flow allocation interface exposed by VTN.

C. Walk-through of Transport Flow Allocation

When an application wants a transport flow with a certain QoS requirement to another application process, it uses the flow allocation interface exposed by its VRM. When the VRM gets the request, it first checks whether any of its available VTNs can reach that application. If the VRM finds such a VTN, it uses the VTN interface to allocate the flow through the transport process belonging to that VTN on the same host. If no VTN is found, the VRM sends the flow request to the VAA of the host, which then forwards the flow request to the VA of the network, and eventually the VA determines how to build a new VTN which consists of new transport processes running on the source host, destination host and some intermediate hosts.

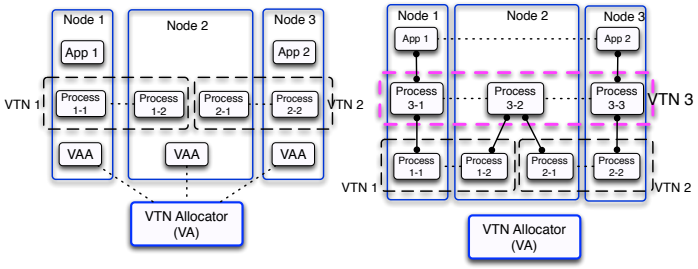


Fig. 6: (a) An enterprise network consists of three hosts (*Node 1*, *Node 2* and *Node 3*), and a centralized VTN Allocator. (b) A new VTN (*VTN 3*) is formed to support the flow between *App 1* and *App 2*. VAA of each node is not shown in (b).

As shown in Figure 6(a), when *App 1* on *Node 1* asks its VRM (not shown) for a flow to *App 2*, and its VRM cannot find an existing VTN to reach *App 2*, *App 1*'s VRM then sends the request to *Node 1*'s VAA, which forwards the request to the VA. The VA figures out that a new VTN is needed to support the flow, then it builds a new VTN (*VTN 3*) spanning all three nodes (Figure 6(b)). After the new VTN is ready, the VAA of *Node 1* notifies the VRM of *App 1*, which eventually uses this new VTN (*VTN 3*) to create a flow to *App 2*. Note than links between processes of *VTN 3* constitute *virtual transport* links and not simply routing tunnels.

IV. Network API

Our management layer provides two sets of APIs: (1) the management-level API, used by network managers to manage the network by programming the VTN Allocator and VTN Manager; and (2) the user-level API, used by regular users to program their own applications, and it helps users affect their application traffic to improve user application performance.

A. Management-level API

Our management-level API includes three sets of APIs as shown in Table I, where the VTN formation API (1) is our new design, and (2) and (3) are existing RINA APIs [7] but reproduced here for completeness.

(1) VTN Formation API
public boolean createVTN (VTNRequest vtnRequest);
public boolean deleteVTN (VTNRequest vtnRequest);
(2) Flow Allocation API
public int allocateFlow(Flow flow);
public boolean deallocateFlow(int handleID);
public void send(int handleID, byte[] msg) throws Exception;
public byte[] receive(int handleID);
(3) Network Information API
public int createEvent(SubscriptionEvent subscriptionEvent);
public boolean deleteEvent(int subscriptionID);
public Object readSub(int subID);
public void writePub(int pubID, byte[] obj);

TABLE I: Three sets of management-level APIs in Java.

1) **VTN Formation API***:* VTN Allocator Agents (VAAs) expose the VTN Formation API (shown in Table I (1)), which is used by the VTN Allocator (VA) to create new VTNs or delete existing VTNs. A VTN is instantiated with different policies such as routing policies, addressing policies and flow and error control policies.

2) **Flow Allocation API***:* The VRM of each application process (cf. Section III-B3) exposes the Flow Allocation API (shown in Table I (2)), which is used to create/delete transport flows with QoS requirements as well as to send/receive data messages over existing flows between management applications.

3) **Network Information API***:* Based on a publish/subscribe model (a pulling mechanism to retrieve information is also supported), the Network Information API (shown in Table I (3)) allows management applications to retrieve or publish network information from/to other management applications. This API (similar to NIB API in Onix [9]) allows management applications to access network information.

B. User-level API

The user-level API includes two sets of APIs: (1) the Flow Allocation API; and (2) the Network Information API. They are the same as the ones in the management-level API.

V. VTN Formation Protocol and Implementation

New VTNs may need to be formed in support of transport flows (Section III-C). In this section, we explain how the VTN Allocator (VA) and VTN Allocator Agent (VAA) interact with each other via a VTN formation protocol to create new VTNs.

A. Objects Exchanged in the Protocol

The key aspect of the VTN formation protocol is two objects exchanged between the VA and VAA.

(1) **Flow Request Object**. The VAA on a host sends a flow request object to the VA when a certain transport flow cannot be supported using existing VTNs on the host. The flow request object specifies the source and destination application information as well as QoS requirements including throughput, delay, and loss rate. The flow request object also supports advanced flow requirements (policies) such as which hosts to bypass or go through, or whether encryption is needed or not. The flow policies inside the request object are specified when the application uses the Flow Allocation API (shown in Table I (2)) exposed by its VRM to allocate the transport flow.

(2) **VTN Request Object**. The VTN request object supports two operations: VTN creation and deletion. A VA sends a VTN request object to multiple VAAs on different hosts once it determines how the new VTN should be formed, $i.e.$, the new VTN should have new transport processes running on which hosts. The VTN request object specifies policies for the new VTN, including policies for routing, addressing, error and flow control, $etc.$ Also it supports other policies such as which application can use this VTN, the lifetime of this VTN as well as resource allocation policies. Furthermore, the VTN request object can specify the connectivity among transport processes, and the enrollment and authentication policies for new transport processes to join the VTN.

B. Protocol Details

A new VTN needs to be formed when existing VTNs cannot satisfy a new transport flow request. Next we explain how the VTN formation protocol works step by step. When the VA receives a flow request object from a VAA, it first inspects the object to see if it is a valid request. If valid, it checks the network state and determines whether there exists a path (a chain of hosts) from the host where the source application runs to the host where the destination application runs. Once a path is found, the VA can decide which hosts the new VTN should span, $i.e.$, the design of the VTN. We call this procedure (including finding a path and designing the new VTN) the *VTN Formation Problem* (details in Section VI).

Once the VA determines the design of the new VTN, it sends the VTN request object specifying the policies of the new VTN to all VAAs of the hosts along the path (including source and destination hosts). When a VAA receives the VTN request, it first inspects the object to see if it is a valid request. If valid, the VAA creates a new transport process as a member of the new VTN, then the VAA sends a VTN response to the VA indicating that the new transport process on this host is ready. After the VA receives the responses from all VAAs to which it sent the VTN request, and if all responses indicate that all new transport processes are ready, the VA sends a flow response to the VAA that sent the initial flow request indicating that a new VTN is ready and the associated VRM can use it to create the transport flow. If any of the VAA's responses indicates failure of the creation of the new transport process, the VA sends a VTN delete request to all other VAAs to delete the newly created transport processes for that VTN. Then the VA sends a negative flow response to the VAA that asked for the new transport flow indicating that the flow request cannot be satisfied.

C. Implementation of VTN-Based Management Layer

We extend the RINA prototype, ProtoRINA [7], to support the dynamic formation and scoped management of VTNs.

We have tested our implementation both on our local campus network and on the GENI testbed [26]. We plan to release our code extension upon acceptance of this paper. Both network managers and regular users can use our implementation to achieve either better network or application performance.

VI. VTN Formation Problem

Next we explain the VTN formation problem for the VA which determines the design of the new VTNs needed to support new transport flow requests, and we also give a practical solution to it. To the best of our knowledge, our paper is the first to formulate the flow allocation problem as a VTN formation problem. In this paper, we focus on solving the VTN formation problem for a batch of flow requests. The online case can be solved either as a special case of the batch case with only one request, or by queuing requests until a certain queue length is reached, then solve the batch case.

Notations for Level-$(n-1)$ Network Topology	
$G_{n-1} = \langle V, E^{n-1} \rangle$	level-$(n-1)$ network topology
$V = \{v_i\}$	set of switches
v_i	switch v_i
$E^{n-1} = \{e_{st}\}$	set of links between switches
e_{st}	(virtual) link between switches s and t
C_{st}	capacity (both directions) of link $e_{st} \in E$
Notations for Flow Request	
$F = \{f_n\}$	set of all flow requests
$N = \|F\|$	total number of flow requests
$b(f_n)$	bandwidth required for flow n
$P(f_n) = \{p_k^n\}$	set of possible paths for flow n
$\|P(f_n)\|$	number of possible paths for flow n
$p(f_n)$	path selected for flow n
$l(p_k^n)$	length of path k for flow n
$F' = \{f_m\}$	set of accepted flow requests after path selection
$\|F'\|$	number of accepted flow requests
$p(f_m)$	path selected to serve flow f_m
$\|p(f_m)\|$	length of the path selected to serve flow f_m
Notations for VTN	
$D^{n-1} = \{D_z\}$	set of existing VTNs of and below level-$(n-1)$
$\|D^{n-1}\|$	number of existing VTNs of and below level-$(n-1)$
$D_z = \langle V_z, E_z \rangle$	VTN z
$V_z = \{v_s^z\}$	set of transport processes in VTN z
v_s^z	transport process s in VTN z (running on some switch v_i)
$E_z = \{e_{st}^z\}$	set of (virtual) links in VTN z
e_{st}^z	(virtual) link between processes s and t in VTN z

TABLE II: Notations for the VTN formation problem.

The notations used are shown in Table II. The VTN (recursive) formation problem has two stages: (1) the path selection stage, where a path on the given level-$(n-1)$ topology[4] is selected for each given flow request; and (2) the VTN design stage, which determines the number of level-n VTNs needed and which hosts (switches) should have transport processes for each VTN. Note that the VTN formation problem can be applied recursively if level-n VTNs cannot satisfy flow requests due to limited scope. Initially, transport over each physical link is provided by a level-0 VTN, which only has two transport processes, one on each end of a physical link. So for the base case of our recursion, the level-0 network topology is given by the virtual links supported by level-0 VTNs and hosts running level-0 VTN transport processes. For the inductive case (level-n, $n \geq 1$), the given topology is a virtual topology, where each link between two hosts is a virtual link, and two hosts have a (virtual) link between them if they have transport processes belonging to a common existing underlying level-$(n-1)$ VTN.

For example, in Figure 6(a), *Node 1* and *Node 3* do not have a (physical or virtual) link between them. The level-0 virtual topology consists of two virtual links: one between *Node 1* and *Node 2* (supported by *VTN 1*), and another between *Node 2* and *Node 3* (supported by *VTN 2*). But *VTN 3* is formed through *Node 2* in Figure 6(b). Figure 6(b) now has two levels of topologies. The level-1 topology has one more virtual link provided by *VTN 3*, and *App 1* and *App 2* can readily use it to communicate.

Our algorithms only focus on two levels of VTNs, $i.e.$, level-$(n-1)$ and level-n, but by recursion we can build as many levels of VTNs as needed. The VTN formation problem is recursively solved when the network manager would like to serve a new batch of flow requests on top of the existing higher-level (virtual) network topology.

A. Stage I: Path Selection Stage

Given a network topology $G_{n-1} =< V, E^{n-1} >$[5] and a total of n flow requests, the goal is to find a path (if possible) for every flow request satisfying its throughput requirement[6]. This path selection can be formulated as an Integer Linear Programming (ILP) problem as shown in Table III. In this problem, $O_k^n = 1$, if $p(f_n) = p_k^n$ ($i.e.$, among all possible paths for f_n, path k is selected); otherwise 0. The objective function is to maximize the number of flow requests served while seeking paths with shorter length (by using the inverse of path length as weight) for each flow as long as the link capacity constraints are satisfied. Line (1) guarantees only one path is selected (if possible) for each flow f_n. In Line (2), $\chi_n^{st} = 1$ if $e_{st} \in p(f_n)$ ($i.e.$, link e_{st} is on the path selected for f_n); otherwise 0. Line (2) guarantees that the bandwidth requirements of all flows going through a link do not exceed the capacity of that link.

Objective: maximize $\sum_{n=0}^{N} \sum_{k=0}^{|P(f_n)|} O_k^n \times \frac{1}{l(p_k^n)}$, such that

$$\forall f_n \in F : \sum_{k=0}^{|P(f_n)|} O_k^n \leq 1 \quad (1)$$

$$\forall e_{st} \in E : \sum_{n=0}^{N} b(f_n) \times \chi_n^{st} \leq C_{st} \quad (2)$$

TABLE III: Path selection formulated as an ILP problem.

We are able to use CPLEX[27] to solve the ILP problem in Table III. Note that some flow requests may not be served due to link capacity constraints, but with aggregation we can satisfy the same QoS requirement with less effective bandwidth usage (cf. Figure 3). Thus after aggregation we have more capacity left and in turn we are able to accept more flow requests. So after solving the current ILP problem, we compute

[4]For $G_n = \langle V, E^n \rangle$, where $n \geq 0$, V is the same for all n since V is the set of switches. E^n represents the set of all virtual links, which grows as we build more (higher-level) VTNs.

[5]In this paper, our path selection is done over the level-0 network topology $G_0 =< V, E^0 >$, which is a one-to-one mapping to the physical topology.

[6]In this paper, we use throughput as an example of QoS support. Other features such as latency and packet loss can be easily considered.

the effective bandwidth usage on each link (cf. Equation 2) and update its link capacity C_{st}. Then we repeat solving the path selection problem in Table III using CPLEX for the unaccepted flow requests based on the new residual link capacity. This procedure is stopped when no more flow requests can be accepted or all flow requests have been accepted.

In the end, the set of all flow requests that can be accepted is denoted as F'. And the accepted flow requests which are mapped on the same level-0 path between the same pair of source and destination switches can be aggregated into one flow.[7] Each $f_m \in F'$ will be then supported by some VTN which is designed in the next stage. In other words, all accepted flows mapped on the same level-0 path are all supported by a single path in some level-n VTN, where each (virtual) link of that VTN is supported by a flow through one level-$(n-1)$ VTN. Through aggregation we can accept more flow requests while reducing the number of forwarding entries installed on the intermediate switches as discussed in Section III-A.

B. Stage II: VTN Design Stage

In this stage, we determine how many new VTNs need to be formed on top of an existing level-$(n-1)$ network topology ($i.e., G_{n-1}$) and existing level-$(n-1)$ VTNs ($i.e., D^{n-1}$), and determine which hosts each new VTN has to span to support each $f_m \in F'$, where F' is the set of flow requests that are accepted after the previous stage. The outputs of this stage are: (1) a set of new VTNs, ($i.e., \{D_z\}$), (2) the host (v_i) that each new transport process in each VTN ($i.e., v_s^z$) is assigned to, and (3) how each new process is connected, $i.e.$, virtual links ($\{e_{st}^z\}$) in each VTN. Note that a new VTN may need to be supported by multiple level-$(n-1)$ VTNs as shown in Figure 6(b). The goal of the VTN design stage is to support all $f_m \in F'$ with the least number of new VTNs (and corresponding new transport processes), which minimizes the required resources as each new transport process in a new VTN consumes a certain amount of host resources such as CPU and memory.

A greedy recursive VTN design algorithm[8] is shown in Algorithm 1. In this algorithm, a VTN is designed to support each $f_m \in F'$ if the VTN has transport processes on each host along the path ($i.e., p(f_m)$), and there is a (virtual) link between processes on each pair of neighboring hosts along the path. So for $f_m \in F'$, the corresponding level-n VTN needs to span all hosts along the path, and the flow is mapped to a path in this level-n VTN, where each link on this path is supported by a flow in one level-$(n-1)$ VTN.

Assume each VTN can have at most M ($M \geq 2$) transport processes. By setting different values for M, we could decide the total number of newly created VTNs as well as their size. Namely, a larger M yields larger management scopes, and a smaller M yields smaller ones. Let us denote $\Delta(f_m, D_j)$ as the number of new transport processes to be added to an exiting VTN D_j so that D_j can support f_m. For example, assume $p(f_m)$ includes 5 hosts, and D_j already has transport processes on 3 of these hosts, then $\Delta(f_m, D_j)$ is 2.

Algorithm 1 VTN_Design $(F', G_{n-1}, D^{n-1}, M)$

```
1: t = |F'|, E^n = E^{n-1}, D^n = D^{n-1}
2: while t > 0 do
3:     get a f_m ∈ F' with largest |p(f_m)|
4:     if any existing D_k in D^n has processes on all v_i ∈ p(f_m) then
5:         add links to D_k and E^n, for each pair of neighbors in p(f_m)
6:         F' = F'/f_m
7:     else
8:         get a D_j ∈ D^n with largest Δ(f_m, D_j)
9:         if (|V_j| + Δ(f_m, D_j)) ≤ M then
10:            for all v_i ∈ p(f_m) do
11:                if no existing process in D_j is assigned to v_i then
12:                    s = |V_j| + 1, V_j = V_j ∪ v_s^j and assign v_s^j to v_i
13:                end if
14:            end for
15:            add links to D_j and E^n, for each pair of neighbors in p(f_m)
16:            F' = F'/f_m
17:        else
18:            if |p(f_m)| ≤ M then
19:                l = |D^n| + 1, V_l = φ
20:                for all v_i ∈ p(f_m) do
21:                    s = |V_l| + 1, V_l = V_l ∪ v_s^l, and assign v_s^l to v_i
22:                end for
23:                add links to D_l and E^n, for each pair of neighbors in p(f_m)
24:                D^n = D^n ∪ D_l, and F' = F'/f_m
25:            else
26:                f_m cannot be supported by the current level
27:            end if
28:        end if
29:    end if
30:    t = t − 1
31: end while
32: Merge_VTN(D^n, M)
33: if F' = φ then
34:    Return G_n =< V, E^n > and D^n
35: else if F' ≠ φ then
36:    VTN_Design (F', G_n, D^n, M)
37: end if
```

We may use an existing VTN (lines 4-6) or expand an existing VTN (lines 8-16) to support a flow, as long as the number of transport processes in that VTN does not exceed M. If we cannot support this flow using existing VTNs, we create a new VTN (lines 19-24). Note that due to the limitation of M (line 18), we may need to build a VTN of even higher level to serve a flow f_m (line 26), and this is done by the recursive call in line 36. As we have more levels of VTNs, we would have more (virtual) links (lines 5, 15 and 23) and yield higher level paths with smaller length. But in this paper, we focus on building only one more level of VTNs ($i.e.$, level-1 VTNs) by choosing an M value that is larger than $|p(f_m)|$ for all f_m (line 18).

At the end of each recursion call, we try to further reduce the number of VTNs by merging existing VTNs in the same level (line 32) using Algorithm 2. Two VTNs can be merged if the number of transport processes in the merged VTN does not exceed M. After this stage, a set of new VTNs is designed, and the VA of the network generates corresponding VTN request objects (cf. Section V-A) and sends them to VAAs to build new VTNs by creating new transport processes on their hosts.

VII. Performance Evaluation

In this section, we present the performance evaluation of our VTN-based management layer.

[7] In practice, we aggregate a set of aggregatable flows if the aggregated flow uses less memory entries than without aggregation.

[8] The VTN design problem can be proved to be NP hard by reduction from the knapsack problem. We skip the proof due to space limitation.

Algorithm 2 Merge_VTN (D^n, M)

$D = D^n, D^n_m = \phi$
while $|D^n_m| \neq |D|$ **do**
  pick a $D_i \in D^n$ where $D_i \notin D^n_m$
  while $|D_i| < M$ **do**
    if $\exists D_j \in D^n$, where $i \neq j$ **and** $D_j \notin D^n_m$ **and** $|V_i \cup V_j| \leq M$ **then**
      $D_i = D_i \cup D_j$, $D^n = D^n / D_j$, $D^n_m = D^n_m \cup D_j$
    else
      break // *exit inner while loop*
    end if
  end while
  $D^n_m = D^n_m \cup D_i$
end while

A. Comparison with SDN-based Solutions

As mentioned in Section III-A, our VTN-based management layer allows aggregation of flows that cannot be aggregated using SDN-based management layers due to distinct IP prefixes and port numbers. Flow aggregation helps achieve better resource utilization to accept more flow requests and reduce memory usage in switches for storing forwarding rules.

We use BRITE [28], a topology generation tool to generate an enterprise network (50 switches and 200 directed links) using the Waxman model ($\alpha = 0.15$ and $\beta = 0.2$), and then randomly generate flow requests between pairs of switches. Each physical link has a capacity of 100 Mbps, and each flow request has an instantaneous traffic demand which follows a uniform distribution between [0, 1] Mbps. Assume the QoS requirement (cf. $1-\epsilon$ in Equations (1) and (2) in Section III-A) for each flow request is 90%.

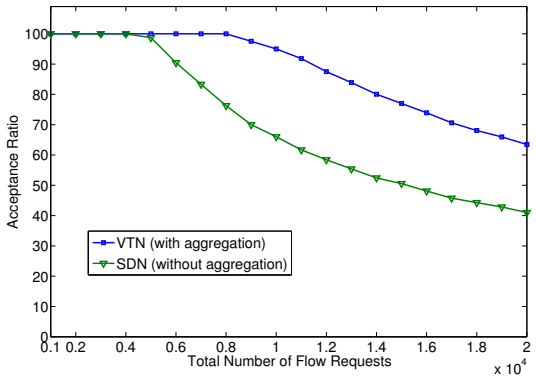


Fig. 7: Acceptance ratio of flow requests, where $1 - \epsilon = 90\%$.

1) **Flow Request Acceptance Ratio***:* Because of aggregation, our management layer is able to satisfy the same QoS requirement with less bandwidth reservation, and thus can accept more flow requests. Figure 7 shows the acceptance ratios for different number of flow requests. When the number of flow requests is less than $5,000$, both VTN and SDN solutions can accept all flow requests ($i.e.$, acceptance ratios are both 100%) since the total traffic demand is less than the network capacity. But as the number of flow requests increases (higher traffic demand), we can see that our solution has a higher acceptance ratio than an SDN-based solution.

2) **Memory Usage***:* Our management layer is able to save memory usage in switches for storing forwarding rules due to aggregation. Figure 8 shows the average number of memory entries needed to serve each flow as the number of flow requests increases. We can see that our VTN-based solution has less per-flow memory usage compared to an SDN-based solution. Also we can see that as the number of flow requests increases, per-flow memory usage decreases for our VTN-based solution. This is because more flows can be aggregated, $i.e.$, the higher the number of flow requests, the better performance our VTN-based management layer achieves. Note that per-flow memory usage for the SDN-based solution (which depends on average path length) also decreases, and this is because the flow requests are randomly generated, and the average path length of flows accepted decreases as we have more flow requests.

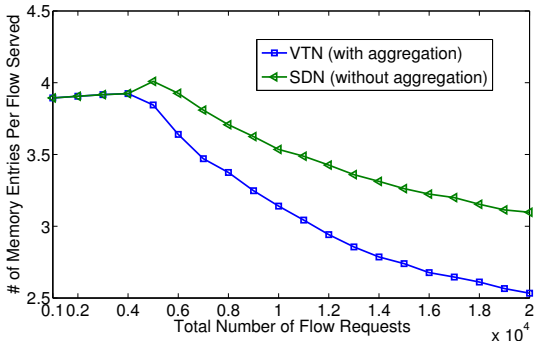


Fig. 8: Average # of memory entries needed for each flow served.

3) **Number of New VTNs***:* Figure 9 shows the average number of new VTNs created per 1000 flows served for different values of M ($i.e.$, maximum number of processes allowed in a VTN) obtained using the VTN design algorithm (Algorithm 1). We can see that as the number of flow requests increases, the number of new VTN needed per 1000 flows served decreases. Also as expected, the higher the number of processes allowed in a new VTN, the less the number of VTNs needed.

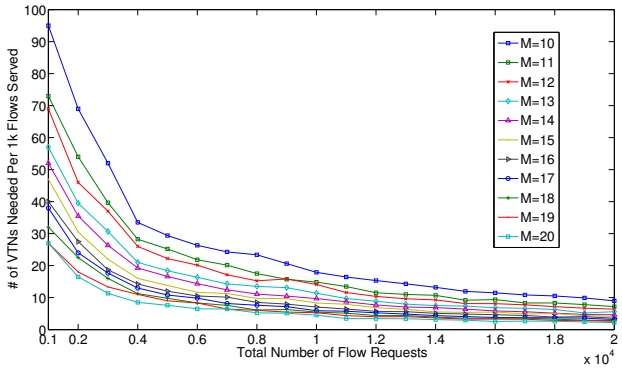


Fig. 9: Number of new VTNs created per 1000 flows served for different values of M.

Due to space limitation, we only show the comparison for a network with 50 switches, but our comparisons for larger networks and more flow requests show similar results.

B. Experiments on the GENI Testbed

GENI [26] is a nationwide infrastructure that allows large-scale networking experiments. In our experiment, we reserve

21 VMs from the Clemson aggregate, and connect them as an enterprise network, which is managed by our VTN-based management layer. We have a centralized VA running on one VM. We define the time needed to build a VTN as the time it takes to successfully create transport processes on all hosts that this VTN needs to span. We measure the time needed to build a new VTN of different sizes assuming this VTN is needed to support flows between different number of pairs of hosts. For each given size of VTN, we run the experiment 5 times, and Figure 10 shows the mean and standard deviation of the time to build the VTN as the size of the new VTN increases from 2 to 20. We observe, as expected, that as the size of the VTN increases, more time is needed. Also the time increases almost linearly with respect to the size of VTN. Practically, we may not always need to create new VTNs from scratch, and we can form a bigger VTN by expanding one of existing VTNs.

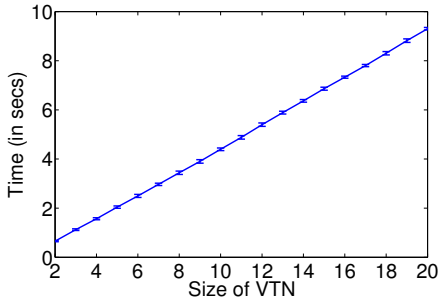


Fig. 10: Time needed to build a VTN of different sizes.

Our current implementation of the VA builds the new VTN in a sequential manner, $i.e.$, the VA sends the VTN request to the VAA of the first host, and only after the transport process is ready on that host, the VA sends the VTN request to the next host until a new transport process is ready on every host needed. Obviously, if the VA sends requests in a concurrent manner, less time will be needed to build the new VTN. Also by creating multiple VTNs in a concurrent manner, less time will be needed to build multiple VTNs following the outputs of the VTN design stage in Section VI-B.

VIII. Conclusion and Future Work

In this paper, we present a recursive application-driven approach to network management in response to the limitations of current SDN-based management solutions. Our approach manages the network using Virtual Transport Networks (VTNs), where each VTN is a virtual network that can provide communication service via transport flows with explicit QoS support. A VTN can be dynamically formed and instantiated with different policies to satisfy different application-specific requirements. We present the design and implementation of a management layer based on our approach, and we also propose a VTN formation problem along with a practical solution, which aims to improve network performance. In the end we show the performance and advantages of our management layer through experimental results.

Our future work includes overlaying our RINA-based implementation over OpenFlow [3] switches. We will further investigate the benefits of using VTNs on resource allocation and application performance. We also plan to explore the VTN formation problem proposed in this paper from an algorithmic perspective.

References

[1] National Institute of Standards and Technology, "The NIST Definition of Cloud Computing," 2011.

[2] Open Networking Foundation White Paper, "Software-Defined Networking: The New Norm for Networks," April, 2012.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM CCR*, vol. 38, no. 2, Mar 2008.

[4] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-deployed Software Defined Wan," *SIGCOMM CCR*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[5] Y. Wang and I. Matta, "SDN Management Layer: Design Requirements and Future Direction," in *CoolSDN 2014*.

[6] J. Day, I. Matta, and K. Mattar, "Networking is IPC: A Guiding Principle to a Better Internet," in *ReArch 2008*.

[7] Y. Wang, I. Matta, F. Esposito, and J. Day, "Introducing ProtoRINA: A Prototype for Programming Recursive-Networking Policies," *ACM SIGCOMM Computer Communication Review (CCR)*, July 2014.

[8] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," *SIGCOMM CCR*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[9] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," in *OSDI 2010*.

[10] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "FlowVisor: A Network Virtualization Layer," in *Technical Report, OpenFlow-TR-2009-1, OpenFlow Consortium*, 2009.

[11] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN Programming Using Algorithmic Policies," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 87–98, Aug. 2013.

[12] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory Networking: An API for Application Control of SDNs," *SIGCOMM CCR*, vol. 43, no. 4, pp. 327–338, aug 2013.

[13] Open Networking Foundation, "https://www.opennetworking.org/."

[14] D. Drutskoy, E. Keller, and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," *Internet Computing, IEEE*, Mar 2013.

[15] E. Salvadori, R. Doriguzzi Corin, A. Broglio, and M. Gerola, "Generalizing Virtual Network Topologies in OpenFlow-Based Networks," in *GLOBECOM 2011*.

[16] X. Jin, J. Gossels, J. Rexford, and D. Walker, "CoVisor: A Compositional Hypervisor for Software-defined Networks," in *NSDI 2015*.

[17] J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker, "Extending SDN to large-scale networks," *OpenNetworkingSummit, 2013*.

[18] The SPARC Project: Split Architecture for Carrier Grade Networks, "http://www.fp7-sparc.eu/."

[19] B. Salisbury, "TCAMs and OpenFlow - What Every SDN Practitioner Must Know," https://www.sdxcentral.com/articles/contributed/sdn-openflow-tcam-need-to-know/2012/07/, 2012.

[20] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-based Networking with DIFANE," in *ACM SIGCOMM 2010*.

[21] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "One Big Switch" Abstraction in Software-defined Networks," in *CoNEXT 2013*.

[22] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *INFOCOM 2013*.

[23] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "Scalable Rule Management for Data Centers," in *NSDI 2013*.

[24] X. N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Optimizing Rules Placement in OpenFlow Networks: Trading Routing for Better Efficiency," in *ACM SIGCOMM HotSDN 2014*, Chicago, USA.

[25] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *SIGCOMM 2004*.

[26] GENI, http://www.geni.net/.

[27] IBM CPLEX Optimizer, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[28] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *MASCOTS 2001*.