SDN AND OVERLAY NETWORK HYBRID SYSTEM FOR FLEXIBLE AND HIGH PERFORMANCE NETWORK VIRTUALIZATION

By

KYUHO JEONG

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2017

To you as a reader

ACKNOWLEDGMENTS

First and foremost, I sincerely thank to my academic advisor, Dr. Renato Figueiredo, who has guided me patiently toward this ultimate goal. Despite of my frequent change of academic topic and deviation from main focus during my PhD study, he enduringly ushered me to concentrate on the right direction. While I got stuck in petty details, he always presented me the big picture and integral insight of the problem and inspired me to come up with solutions. He not only helped me academically but also gave insightful lessons of life such as the importance of social and writing skills to interact with other insightful researchers, and offered opportunities to improve this with the various projects. During my PhD study, I have worked with researchers and developers around the world such as the universities and institutes in Italy, Netherland, Miami, San Diego, China and Japan. I would not have these valuable experiences in this short period of time without his drive for interdisciplinary and eager for synergy with other researchers.

I also show my gratitude to Dr. Jose Fortes, Dr. Xiolin Li and Dr. Shigang Chen for being my committee members, providing insightful inputs toward my research, and encouraging me to challenge daunting research topic.

I express my gratitude to Dr. Kohei Ichikawa, whom I always call for help when I had short of insight in computer network field. Had not been the help from him, my research would be far below level. Even the time difference and long distance could not prevent his helping nature, he always responds kindly with insightful ideas upon my call for help.

I appreciate Dr. Pierre St Juste, Benjamin Woodruff and Dr. Heugnsik Eom who left the legacy of IPOP project from which I can learn and extend my research. Were it not their endeavor of developing, careful design decision, and quality assurance for the reliable software project, I could not extend my insights on overlay network. I did not acknowledge your importance at the time you were around me. But the sense of void I felt after you left cannot help but appreciate your presence in the early of my PhD study.

As I am in completing my degree program, I cannot help but remind the conversation with Dr. Ming Zhao when we met in Kyoto, Japan. When I felt lost in pursuing my path, he encouraged me to set an ambitious objective to drive myself further. I thank for his nudge.

I already feel nostalgia of good old days in ACIS with Dr. Yonggang Liu and Dr. Kyungyong Lee. I thank for hanging around with me in leisure time and extending my knowledge from their research. And both of your successes after the graduate motivated me for pursuing the PhD study.

It is hard to show my gratitude enough for Giljae Lee, Kyungmin Park and Dr. Jongmin Lee. As the mentors of life and research, they advised me, spend time with me, and listened patiently to me.

Finally, I especially thank to my family. I feel remorse all the time about not being with you often. No matter how long I think, I cannot find a correct enough rhetoric that expresses your devotions to me. So, I plainly state that I sincerely thank for all of your devotion.

This material is based upon work supported in part by the National Science Foundation under Grants No. 1527415, 1339737 and 1234983, and by the Department of Defense award W911NF-13-1-0157. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the Department of Defense. The author acknowledges the use of NSFCloud resources such as CloudLab and Chameleon for the experiments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

SDN AND OVERLAY NETWORK HYBRID SYSTEM FOR FLEXIBLE AND HIGH PERFORMANCE NETWORK VIRTUALIZATION

By

Kyuho Jeong

December 2017

Chair: Renato J. Figueiredo
Major: Electrical and Computer Engineering

Network protocols and infrastructure are difficult to be modified after deployment. Thus, overlay networks have been widely used to provide a virtual abstraction of network layer, achieved by encapsulation at the end-points and tunneling through the network fabric. Although overlay networks provide the flexibility to implement operations such as modifying/mapping identifiers and control of information for network virtualization, they are not able to keep the pace with the transmission rate capacity of modern hardware and communication links, and consume computing resources from the end-hosts. The recent emergence of Software Defined Networking (SDN) created a means of virtualizing network from infrastructure by using common APIs. This dissertation addresses challenges in the context of overlay network design, including performance improvements by using SDN in the contexts of inter-cloud and multi-tenant data centers.

In the context of inter-cloud architectures, this dissertation investigates a novel approach called VIAS (VIrtualization Acceleration using SDN) to the design of overlay networks. VIAS uses SDN to selectively bypass tunneled links when SDN-managed paths are feasible. Architecturally, VIAS is self-organizing, whereby overlay nodes can detect that peer endpoints are in the same network and program bypass flows between OpenFlow switches.

In the context of multi-tenant data centers, the virtualization performance of overlay network technologies such as VXLAN is far below link speeds, and overlay processing consumes substantial resources from the hypervisor. This dissertation introduces PARES (PAcket

REwriting on SDN), which uses the packet rewriting feature of SDN switches to enable multi-tenant functionality with near-native performance and reduced load on end-hosts.

In public WAN Internet, along with the emergence of cloud computing, overlay networks can be used to increase end-to-end bandwidth by using additional cloud paths, instead of using only the default Internet path. This dissertation empirically evaluates the extent to which using cloud paths to transfer data in parallel with the default Internet path can improve the end-to-end bandwidth in bulk data transfers.

CHAPTER 1
INTRODUCTION

Network protocols cannot be modified without significant effort and cost after they become widely adopted. To address this issue, overlay networks have been used in several scenarios to provide flexibility on top of an existing unmodified infrastructure [3, 36, 4]

For example, a user may want to create a Virtual Private Network (VPN) [4] connection between two nodes separated by multiple Network Address Translation (NAT) [83] devices across public Wide Area Network (WAN). She may choose one of many known overlay network technologies (such as IPsec [48], GRE [36] or IPOP [47]), which capture packets or frames from the end-points and encapsulate with another network header such (as GRE or UDP). Then, these outer network headers enable the packets to traverse the fabric based on the outer address, then to reach to the destination end points.

As another example, in a virtualized multi-tenant data center, the administrator wants to decouple the actual network identifier of physical network interfaces used by hypervisors from the virtual network interfaces of tenants' virtual machines. Thus, she may choose one of many overlay networks technologies (such as VXLAN [55] or NVGRE [26]), which encapsulates tenant's packets by the hypervisor network identifier to build arbitrary virtual network topologies and traverse the packets across the data center [43].

Another use of overlay networks is to add supplemental paths to increase aggregate bandwidth or reliability. Internet routing path between a given source and destination pair is determined by protocols between Autonomous System (AS) [37] and then commonly by Classless Inter-Domain Routing (CIDR) [25]. However, the path is not always the best path in terms of bandwidth and packet loss rate. A user may want to use a path other than the default Internet path; however, this is often not practical, not only because end-hosts do not have authority to modify the routing rule, but also because the decision is made by multiple entities such as ASes. Alternatively, a user can intentionally deploy additional cloud nodes across the public WAN, and use an overlay network approach to support packets to be routed

through the cloud-provisioned nodes. Throughout this dissertation, we refer to this alternate path using the above technique as cloud-path.

As stated above, overlay networks have been widely used in various scenarios. However, overlay networks have inherent overheads including encapsulation and context switching. Moreover, although overlays can provide additional flexibility and additional network resources, widely used protocols and APIs are not designed to expose overlay functionality.

This dissertation studies the above three usage scenarios of overlay networks. In the remaining of Chapter 1, I briefly survey overlay networks, Software Defined Networking (SDN) [57], Multi-Tenant Data Center (MTDC) [61] architectures, and cloud-routed overlay path (cloud-path) [11], as those are the foundational concepts of the problem domain this dissertation deals with. Chapter 2, 3 and 4 are dedicated to studies and findings of the usage scenarios stated above respectively.

1.1 Overlay Paradigm

The philosophy of IP-based network is to keep the middle simple and, the edge smart. Best-effort packet delivery in network layer is characterized as unreliable and connectionless. Thus, packets may be corrupted, lost, or delivered out-of-order. What makes possible to enforce reliable delivery end-to-end are protocols at the transport layer, which are implemented above IP layer, on edge/end devices.

However, IPv4 address exhaustion and the necessity for isolation from the public Internet for security, inevitably led to the introduction of Network Address Translation (NAT) at edge network devices, which separates IPv4 address subnet and prevents packets of connectionless protocols to traverse through. In addition, middleboxes (such as load-balancer or proxies), transform transport headers between the end-points. Thus, current network infrastructure is not pure IP network fabric from end to end, but rather a more complex fabric subject to the introduction of network devices such as gateways, NATs, firewalls, or middleboxes.

Since overlay places nodes on the middle of fabric and functions as more than a network/switches devices, the overlay network paradigm contrasts with this "keep the middle

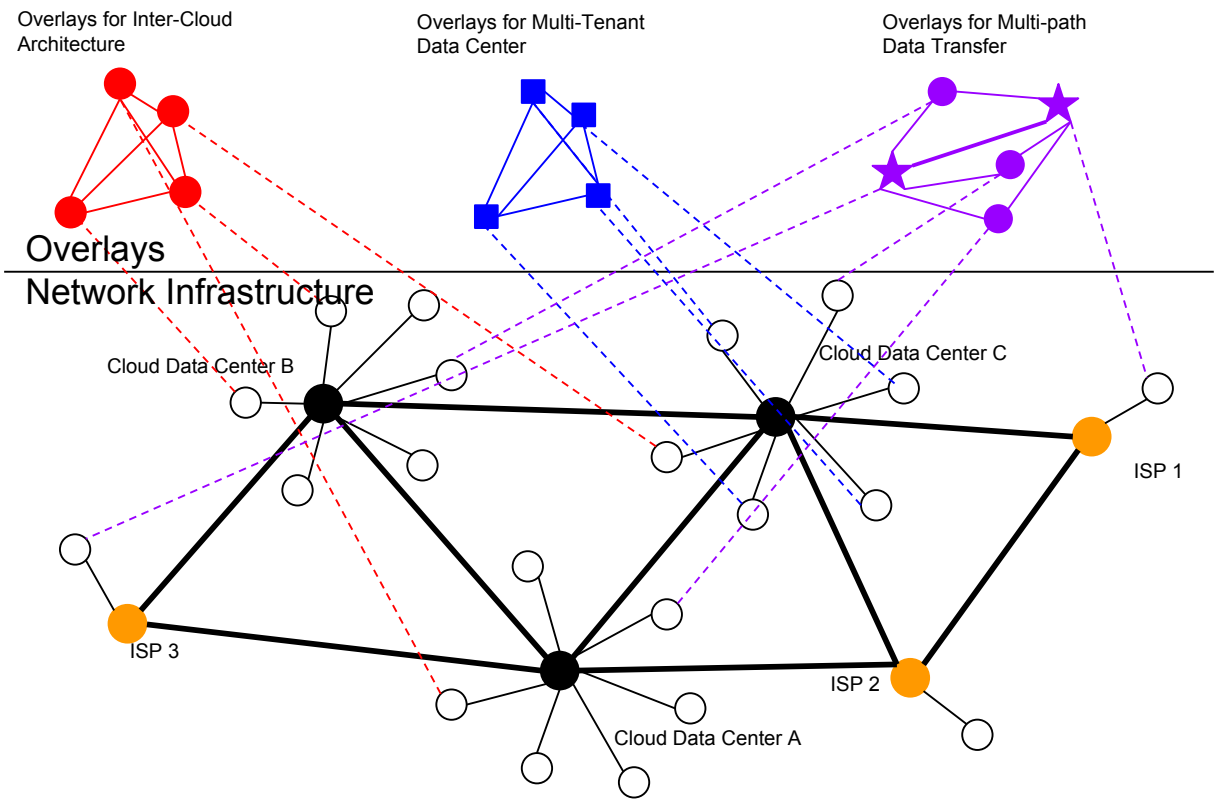

Figure 1-1. Various overlay use cases this dissertation studies

simple and make edge smart" design philosophy. Overlays may place nodes on the fabric, such that they are able to modifiy the packet to meet end-points application interests. For example, RON [3] and X-Bone [88] improve the resilience and flexibility. Yoid [24] and Bullet [52] deploy overlay nodes in the network and allow multicast from a single source to multiple destinations. Chord [85] deploys overlay nodes in a ring-shaped topology and packets are routed based on identifiers to implement systems such as a Distributed Hash Table (DHT).

As such, overlays are now essential in current network infrastructure. In this dissertation, we study three different overlay usage scenarios as shwon in Figure 1-1, one in inter-cloud architecture (as red circle), the other in multi-tenant data center (as blue square) and another in public Internet environment (as purple star). In inter-cloud architecture, overlays are deployed in multiple and different cloud service providers, and provides the layer 2 or 3 abstraction to the VMs as if they are in the same LAN or CIDR environment. In multi-tenant

data center (MTDC), overlays are deployed inside data center and run on the hypervisor to provide the same virtual network environment to the multiple instances of VMs of a tenant. Different from other overlay environment, MTDC requires performance equivalent to line rate, thus the conventional overlay implementation for MTDC requires substantial computing resources from the hypervisor. In final usage case, multiple nodes are deployed in cloud, and end-hosts use these nodes to maximize throughput by aggregating bandwidth. In this case, cloud nodes are used to bypass congested routing path and to diversity available paths, of which the end-users cannot control. Thus, overlays are used as a means to bypass exterior gateway protocols.

1.2 Overlay Networks

Before the emergence of Software Defined Networking (SDN), there were two general approaches to network virtualization. One is through encapsulation, also known as tunneling, to form overlay networks [3, 47]. The other approach is to use an additional protocol, such as VLAN tagging, along with network devices supporting the protocol [55]. The two approaches have advantages and shortcomings. Encapsulation has the advantage of flexibility, since it can virtualize any layer of network frame or packet. However, it needs to delimit the packet/frame size to evade the defragmentation problem. Another drawback is the overhead of appending an additional network header to perform encapsulation. During the encapsulation process, upon all the packets/frames in datapath, a new network header needs to be built along with its new checksum. While hardware network devices use Ternary Content Addressable Memory (TCAM) to achieve packet switching/routing of 10+ Gbps bandwidth on prefix routing [1, 68], with extremely limited number of entries in TCAM, the encapsulation process typically depends on a commodity CPU, which makes it difficult to process packets on par with hardware switches [77]. Besides, applications using encapsulation expose the packets to untrusted public Internet devices/links, requiring encryption, which places additional burden on end-host CPU.

While the latter approach (separate protocol with hardware implementation) has the advantage of line-speed virtualization, it needs to be supported by the network devices. It

also has the disadvantage that a network administrator needs to thoroughly configure all the involved network devices. Furthermore, it may not scale because of limited support from the protocol itself or hardware device implementation. For instance, a VLAN tag identifier has only a 12-bit field, limiting the number of entries to 4096 [55]. Furthermore, the IP multicast entry of commodity routers or switches is of the order of 100s to 1000s, limited by hardware specification. Those limitations in protocol and hardware implementation place scalability barrier upon network virtualization. In addition, switch and router console interfaces are different from vendor to vendor making it harder to manage virtualized systems at scale.

Overlay networks provide a flexible foundation for Virtual Private Networking (VPN), by tunneling virtual network traffic through private, authenticated end-to-end overlay links. Overlay networks have been widely used as a solution for network virtualization at different layers (link, network, and application). The idea of building user-level virtual networks for grid/cloud computing dates back to systems including RON [3], Violin [46], VNET [86], IPOP [47], ViNe [89], hSwitch [22], X-Bone [88], and MBone [19]. RON [3] showed that an application layer overlay on top of Internet routing substrate substantially improves the recovery rate from infrastructure outages. Violin [46] proposed overlay virtual networks providing the abstraction of separate layer-2 networks for tenants. Violin was implemented without requiring any modifications on the VMM (Virtual Machine Monitor) or the hosting network infrastructure, not only providing flexible user configurable network environment, but also reducing the threat of security risk from the host. VNET [86] addressed a similar problem, focusing on the ability to inter-connect virtual machines across providers. This was accomplished by running VNET "proxies" on endpoints (e.g. VMs) at different sites, and tunneling L2 traffic over TCP/TLS links. hSwitch [22] forms an overlay network by creating Generic Routing Encapsulation (GRE) [36] tunnels across dispersed nested VMs. However, because it creates tunnels based on provided configuration files, the overlay topology is static, and dynamic joining/leaving of instances is complex to manage. ViNe [89] can be considered as NFV (Network Functions Virtualization) node. ViNe runs a Virtual Router

(VR), implemented as a form of virtual machine, in a cluster environment, and provides virtual network environment across geo-distributed clusters. Internet Protocol Security (IPSec) is widely used in VPN since it provides authentication, encryption and host to host virtualization. The IP-over-P2P overlay (IPOP) [47] supports both layer 2 and 3 virtual private networking with peer-to-peer tunnels that are dynamically created and managed – even when nodes are behind NATs. Tunnels are created to reflect relationships stored in messaging or online social network services (e.g. an XMPP server), supporting overlay topologies such as unstructured social graphs and structured P2P networks [85].

Many kernel-level overlays are also used, especially in data center. VMware's NSX and Microsoft's Hyper-V run VXLAN [55] and NVGRE [26] respectively on virtual machine monitors and encapsulate packets from virtual network interfaces of guest machines by UDP and GRE header respectively.

While providing flexibility, both user- and kernel- level overlays inherently incur network virtualization overheads, such as user/kernel boundary crossing and header encapsulation, and management overhead, such as virtual network interfaces to attach/detach physical network infrastructure, and setup and management of the overlay itself. Violin, RON and IPOP create their own routing header appended before conventional network routing header, which allows dynamic topology of overlay network and convenient configuration. VNET and hSwitch takes advantage of conventional protocol such as TCP/UDP and GRE respectively as an encapsulation header. VNET achieves relatively high performance virtualization compared to other researches by taking advantage of kernel implementation and multi-thread packet handling [95]. However, the nature of commodity processor handling virtualization process does not change.

In this dissertation, IPOP is used as one of the studied substrates for overlay networking. Although IPOP is not a performance-centric overlay network, it has embedded NAT traversal technique to create a dynamic peer-to-peer tunnel, and has autonomic topology management. In particular, it separates the topology policy module from the encapsulation and encryption

module, separates data plane from control plane, and exposes APIs. Those features make IPOP suitable software to integrate with SDN framework. IPOP works in live migration scenarios, and has been used in the Kangaroo [75] platform-as-a-service system. However, its overlay processing hindered the performance of nested VMs across hosts. This dissertation directly addresses this performance limitation.

1.3 Software Defined Networking

While overlay networks provide network virtualization at the end-points, another approach to deliver network virtualization is from the network infrastructure fabric, along with dedicated protocols. A well-known and widely-used technique, VLAN, keeps layer 2 boundary based on 12-bit field VID (VLAN Identifier). A network administrator can manage layer 2 virtualization by configuring VLAN in network devices, creating virtual layer 2 fabric upon a physical switch fabric. VLAN uses a flat 12-bit identifier to virtualize layer 2 networks, thus it inherently cannot scale to more than 4096 virtual layer 2 networks [55]. Koponen et al. also points out that conventional techniques such as VLANs are not able to cope with the need for flexible configuration and isolation across large numbers of tenant address spaces [51].

Software-defined networking (SDN) [35, 57] has emerged as a general approach to address such challenges, SDN allows network administrators to programmatically and dynamically configure, control, and manage network behavior through open interfaces, and provides a means to monitor traffics of network devices. SDN differs from protocol-oriented network virtualization (e.g. VLAN), as it lets controller define the behavior of switches through open API interfaces. SDN has emerged to provide flexibility in petrified hardware-based network infrastructure, and to unify interfaces among vendors. It divides control and data plane in infrastructure layer, which enables a network operator or administrator to programmatically configure its network infrastructure. Although configuration of network devices was possible even before SDN, the scope of configurability and functionality differs from vendor to vendor. As data center scales out not only in terms of number of physical devices inside the site but

also in terms of number of virtual entities, the level of heterogeneity among network devices increases, which motivates to abstract away network device functionality as a form of SDN.

Although, in SDN, the abstraction of infrastructure is provided to the control plane, allowing even experimental protocols to run on SDN infrastructure, the control plane does not have complete control of the data plane. For example, the dominant SDN protocol, OpenFlow [57], allows match/rewrite of a limited set of fields of the network protocol header, such as MAC/IP addresses, and transport port numbers. However, features such as overall packet rewriting, fragmenting, encrypting, or merging packets or frames, are not part of the OpenFlow specification.

One of the motivations of SDN is the necessity of testing experimental protocols to overcome the difficulties of deploying new protocols on legacy hardware switches and routers [57]. Nonetheless, many researches have been conducted to use SDN as an technique to virtualize the network from the infrastructure service providers' point of view.

VirtualWire [93] takes advantage of connect/disconnect primitives, which are assumed to be exposed by the cloud service provider, and use these primitives to live migrate VMs from one cloud to another cloud. Network Virtualization Platform (NVP) [51] provides REST APIs to the tenants to expose network virtualization capabilities. Network elements such as switches and ports are presented to the tenants, and tenants build the topology of their network. Then, tunnels and flow rules are created and programmed by NVP to each hardware and software OpenFlow switches to forward packets among VMs deployed intra- and inter- clouds. However, NVP is designed to support multiple tenants in a single cloud provider. WL2 [14] presents SDN-based solution to an Internet-scale virtual layer 2 across multiple data centers. It eliminates layer 2 broadcasting by rerouting and soliciting layer 2 traffic in the control plane, and introduces virtual MAC address, which is a concatenation of data center, switch, VM, and tenant identifier, to map flat addresses to hierarchical network entities.

The advent of SDN techniques and the OpenFlow standard have unlocked the potential to address the limitations of virtualization performance, and to address range scalability,

while still deploying network virtualization services at scale within or beyond service providers. However, while SDN-based solutions are becoming well-understood within the context of a single provider's infrastructure, SDN solutions across non-cooperating providers (e.g. different public/private clouds) are currently not feasible, as providers are not willing to allow external entities to exert control in their SDN infrastructure.

1.4 Network Functions in SDN, NFV and Middleboxes

While SDN emerges from the academia, industry and open source community, Network Function Virtualization (NFV) arises from the telecommunication and infrastructure vendors. NFV virtualizes network node functions, typically in the form of virtual machines attached to the network (also known as middleboxes), for improved communication services. Hundreds of telecommunication companies and network device manufacturers are trying to generalize what functions to virtualize, and to identify common software design patterns from what has previously been implemented as a proprietary hardware appliance [12, 56, 42].

Along with data center and server virtualization trends, SDN/NFV now are understood as a key for improving virtualization performance. Subsequently, the need for efficiently supporting large virtual private network to tenants in enterprise data centers, such as public and private clouds, has motivated the adoption of SDN/NFV for data center network virtualization.

In this section, the implementation complexity of various network functions is surveyed and the boundaries between SDN, NFV and middleboxes are clarified. Middleboxes are usually a single entity which is capable of one of various of network functions. Network functions range from load balancer, NAT or proxies in its simplest form [12], to transport-layer encryption (TLS) [64] and Hadoop-layer data aggregation [56], which requires read and write of the whole packet payload.

Figure 1-2 illustrates the complexity of implementing network functions and boundaries of these entities. Each axis shows the required degree of depth in write/read access of packets. As the depth increases, the functions require deeper inspection/write on the packets. The leftmost network function is VLAN, which only requires Layer 2 shallow packet inspection

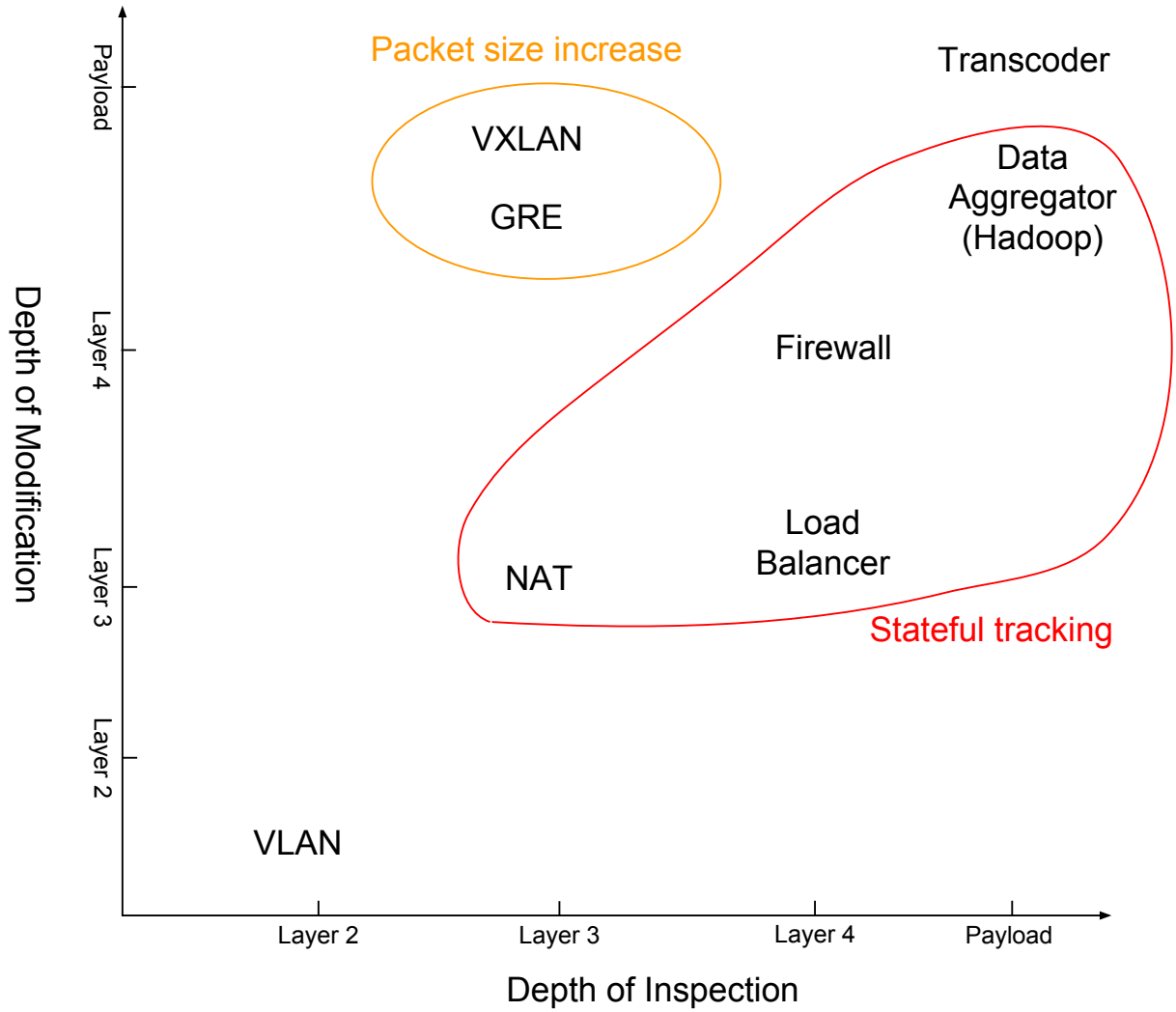

Figure 1-2. Approximate implementation complexity of network functions

and does not require modification of any layer of network header. As it increases horizontally, NFV requires access to higher network layer or deeper in the payload. As such, as depth of inspection increases, network functions require payload inspection such as transcoder (on-the-fly encryption/decryption middlebox for application layer [12]) or Hadoop data aggregator. The vertical axis represents the degree of complexity of rewriting a packet. As it increases, network function requires more complex writing, such as encapsulation or payload rewriting. Simple load balancer can be implemented using rerouting, without the requirement of payload access, while complex HTTP load balancer requires to access payload

to parse HTTP. Note that Figure 1-2 is for visual illustration; stateless and stateful process are arbitrarily placed on the dimension of vertical axis.

While recent network functions require deeper rewrite of packet payload, OpenFlow protocol only allows packet rewriting of a fraction of fields of the network header. From the OpenFlow version 1.5.1, the "Set-Field" action is categorized to "optional" action instead of "required" action, leading to hardware OpenFlow vendors only implementing this action in the slow-path (using a general purpose processor) rather than in the fast-path. This is also known as forwarding and control element separation [63, 50].

Although it is conceivably possible to implement any form of network functions in OpenFlow, by calling into the controller, many are impractical since it involves forwarding whole packets to the controller, applying modification, and sending back to original switches. Thus, in this dissertation, we concentrate on network functions that are implementable in current OpenFlow specification, in the fast when possible, or in the slow-path.

1.5 Network Virtualization in Multi-Tenant Data Center

Multi-tenant data centers for cloud computing require the deployment of virtual private networks for tenants in an on-demand manner, providing isolation and security between tenants. To address these requirements, hypervisors uses protocols such as VXLAN [55] or NVGRE [26]. These protocols run in a kernel layer of hypervisor, encapsulate packets from the guest machines, and tunnels the packet inside data center in secure and flexible manner. However, these approaches inherently incur processing overhead on hypervisor as other user-space overlays do, reducing the effective throughput for the tenant virtual network compared to the native network. Unlike other overlay networks for VPN such as GRE [36] or IPsec [48], where the virtualization process overhead on user-space can keep the pace with the throughput. However, the virtualization process overhead is exacerbated in data center network environment where 10Gbps bandwidth are common.

As server virtualization technologies mature, multi-tenant cloud computing has become widely used as a platform for on-demand resource provisioning. Dynamic provisioning of

computing resources as virtual machines provides flexible computing infrastructure that can be tailored by tenants, and provides high availability and proximity to clients by geographically distributing servers across different zones.

From a networking perspective, multi-tenancy in data centers requires an ability to deploy arbitrary virtual network topologies upon physical network infrastructure, along with managing overlapping addresses between different tenants. In addition, for better utilization and high availability, VM migration is an essential feature in MTDC. When VMs are created upon user's request, they should be virtually connected to the user's already deployed VMs, regardless of their physical location. Also, when live migrated, the network identity (such as MAC or IP address) of VMs should remain the same. Finally, when VMs are terminated, their network presence should be revoked from the tenant virtual network, and network identities should be reclaimed. Tunneling has been widely used to tackle these issues by providing a virtual overlay network upon the physical data center network. Tunneling advantages include ease of deployment and separation from physical network topology, because it obviates the need for additional protocol from physical switches such as VLAN or MPLS (Multiprotocol Label Switching Architecture) [78].

Generally, the tunneling process involves encapsulating every packet from the virtual overlay network by a physical network header. This process is usually achieved by general purpose processors at end-points and implemented in software (e.g. hypervisors), instead of physical network fabric. Nowadays, link speed of 10 Gbps is prevalent, and the trend is toward higher rates. While network devices surpass 10 Gbps, it is increasingly difficult for the tunneling process to keep this pace: although current CPU technology achieves few hundreds of GIPS(Giga Instructions per Second), the processing time of packet classification is dominated by memory access time rather than CPU cycle time [31].

Moreover, the tunneling process hogs up computing resources at hypervisors, nibbling available resources for the guest machines, as encapsulation process includes process of

prepending additional header on every packets and frequent context switch between hypervisor and guest machines.

In Chapter 3, PARES(PAcket REwriting on SDN) is introduced, a novel technique which uses the packet rewriting feature of SDN switches to provide multi-tenancy in data center networks at edge switches, thereby reducing the load on end-point hypervisors and improving the throughput, compared to tunneling. Experiments in an SDN testbed show that the proposed data center architecture with PARES achieves near line-rate multi-tenancy virtualization with 10Gbps links (compared to 20% of line-rate for VXLAN tunneling), without incurring processing overhead at end-point hypervisors or guest servers. Additionally, it evaluates the scalability of PARES for ARP protocol handling and with respect to number of SDN flow entries.

1.6 Cloud-Routed Overlay Path

It is known that the Internet is a loosely-coupled aggregate of multiple ASes (Autonomous Systems), where no single entity has complete authority nor control to optimize congestion or utilization of traffic across the network. This nature of the Internet architecture occasionally leads to a counter-intuitive phenomenon, where a direct shortest route with low latency delivers less bandwidth than alternative, long and roundabout routes. In other words, congestion control and utilization are not globally optimized.

Although ISP-provided dedicated private leased lines can be used for higher bandwidth and improved reliability, line lease contracts occur at the granularity of only a few per year [44], making it practically impossible to use it for a short-term period, or to address sudden spikes in network traffic demand.

Previous research has shown that geospatially distributed computing instances in commercial clouds offer users an opportunity to deploy relay points to detour potentially congested ASes, and as a mean to diversify paths to increase overall bandwidth and reliability. The existence of a variety of commercial cloud computing offerings allows users to swiftly deploy public routable computing resources in a flexible, and distributed manner: users can

deploy VM instances in minutes, and at different geographical locations within the scope of a single provider (different availability zones), or across providers. For example, just within the continental United States, Amazon EC2 operates data centers at four sites (Oregon, Ohio, Virginia and California); other cloud providers have similar geospatial diversity of their data center locations. This geospatial diversity presents a means to use cloud services as a detour relay point to create additional end-to-end paths. CRONets [11] found that 78% of the Internet paths can achieve improved throughput by leveraging alternate cloud-routed paths, and that the improvement factor is about 3 times, on average.

Such opportunity comes with a cost, as cloud-routed paths incur cost of not only provisioning of computing resources, but also for additional traffic to/from Internet. In Chapter 4, it is proved in empirically that the evaluation in the extent to which using cloud paths to transfer data in parallel with the default Internet path can improve the end-to-end bandwidth in bulk data transfers. In the evaluation, we consider single-stream and multi-stream TCP transfers across one or more paths transfers across one or more paths. Moreover, we suggest an application level design pattern that takes advantage of this improved aggregate bandwidth to reduce data transfer times.

1.7 Parallel TCP

Previous work has also shown that parallel TCP streams in a single path can be applied to increase overall throughput [32, 33, 34]. Parallel single-path TCP is widely used in applications such as GridFTP [2], in particular in scientific projects that require bulk data transfers such as Atlas or CERN. However, although parallel TCP is effective in underutilized network environments, if the network is fully utilized, parallel TCP unfairly oust competing TCP streams. Moreover, end-points in public WAN do not have much options to diversity their routing path. As such, parallel TCP expects only a single routing path, as with other transport protocols, but it is not an effective solution to cope with fully utilized and bottlenecked network environments. Nonetheless, the availability of commercial cloud instances on the

Internet, and of various overlay technologies, present options for diversifying routing path on public WAN, which can overcome the limits of a single routing path.

In this dissertation, we use the term bandwidth to refer to the transfer rate of one or more individual streams at the transport layer (e.g. as measured by "iperf"), where the byte stream is not ordered across streams, and throughput to refer to the end-to-end bulk data at the application layer, where the byte stream is ordered at the application layer. In Chapter 4, we show how much aggregate bandwidth can be achieved in bulk data transfers, by concurrently running parallel TCP streams on a default Internet path, along with multiple cloud-routed overlay paths in a public WAN environment. It is observed that that, in many scenarios, additional cloud paths with parallel TCP streams achieved aggregate throughput equal to or greater than the default Internet path (single-stream and multi-stream TCP), while in some cases achieving significantly larger aggregate throughput for end-to-end bulk data transfers. In addition, a design pattern as a pseudo socket API that can leverage the increased aggregate bandwidth of cloud multi-paths to increase overall throughput is presented.

1.8 Dissertation Organization

In Chapter 2, we study hybrid model of overlay networks and SDN called VIAS. VIAS integrates overlay and SDN techniques to support flexible and high-performance virtual networking, in particular across tenant-managed nested virtualization instances distributed across cloud providers. In Chapter 3, we investigate how overlay networks help to provide multi-tenancy in current multi-tenant data center (MTDC), then study and present an SDN-based approach that operates at edge switches of a data center fabric. In Chapter 4, empirical experiment results of increased aggregate bandwidth using cloud-path are described and analyzed. Then, we investigate and implement the socket APIs to leverage this multi-stream aggregate bandwidth and to provide ordered byte stream to the application layer.

CHAPTER 2
OVERLAY AND SDN HYBRID MODEL FOR INTER-CLOUD ARCHITECTURE

Chapter 2 presents VIAS, a novel approach that integrates overlay and SDN techniques to support flexible and high-performance virtual networking, in particular across tenant-managed nested virtualization instances distributed across cloud providers. It does so while exposing a network layer 2 abstraction to endpoints. VIAS supports the general case (the abstraction of a layer-2 virtual private networking linking instances across providers) by employing overlay networking as the substrate for virtualization, and optimizes for a common case (network flows among nodes within the same provider) by means of a novel performance enhancement technique of automatically detecting and programming fast bypath TCP/UDP flows using SDN APIs. Such SDN-programmed VIAS bypass removes the necessity of packet encapsulation and delivers virtualization performance near wire link speed. VIAS detects traffic-intensive TCP/UDP flows inside the overlay-encapsulated data traffic, and automatically switches over to the SDN fabric whenever such path can be programmed – such as when endpoints are within the same cloud provider.

The main contribution of Chapter 2 is a novel system that integrates distributed overlay network and SDN controllers to self-configure peer-to-peer virtual private overlay network tunnels across cloud providers, and transparently detect and program virtual network bypath flows within a provider. The system has been implemented by leveraging an existing overlay technique (IPOP [47]) as a substrate, and integrating existing SDN programmable switches (Open vSwitch) to establish bypass flows. While it can generalize to different overlay tunneling and SDN targets, VIAS applies, in particular, to inter-cloud nested virtualization environments, where tenants can deploy software SDN switches (e.g. Open vSwitch) in their own instances. Chapter 2 demonstrates the functionality of VIAS within and across clouds, and evaluates the performance of the prototype in multiple cloud environments.

The rest of Chapter 2 is organized as follows: Section 2.1 elaborates on the various aspect of network virtualization and environment of inter-cloud architecture. Section 2.3

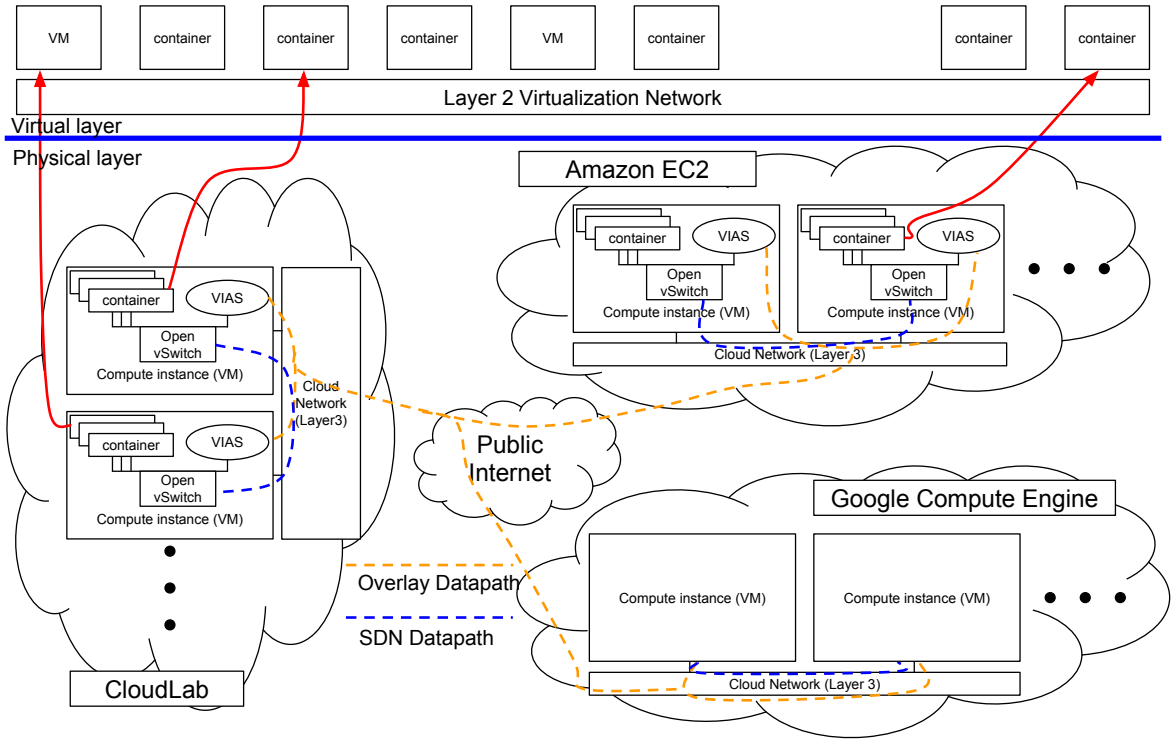

Figure 2-1. Inter-cloud architecture

presents an overview of overlay and SDN hybrid model. Section 2.4 describes the general architecture of VIAS. Section 2.5 provides details on the design and implementation of a VIAS system. Section 2.6 presents results from the evaluation of the prototype in realistic cloud environments. Section 2.7 discusses related work and literature survey, and Section 2.8 concludes this.

2.1 Background

This section overviews nested virtualization, inter-cloud architecture, process containers and virtual networking issues and challenges that serve as a basis to motivate the design and uses of VIAS.

2.1.1 Inter-Cloud Architecture

As cloud computing technology emerges, concept of computing resource changes as a form of utility such as electricity or water. These resources are now provisioned on-demand. And they are provided as a form of virtual machine in a diverse granularity. For example,

"m1.medium" as in Amazon EC2 and medium or n1-standard-1 in Google Cloud Platform. These changes conceptually detach computing resources from physical infrastructure. These days, users deploy their multi-tenant application on provided computing resources from clouds, not on physical servers. Along with these trend, so called inter-cloud architecture emerges, which deploy multi-tenant software upon heterogeneous computing resources across different cloud service providers [29]. For example, single multi-tenant application can be deployed across Amazon EC2, Google compute engine and CloudLab [16] as in Figure 2-1. In this example, user deployed multiple cloud instances across heterogeneous cloud service provider, where using the public Internet as a substrate network and where providing the same layer 2 network abstraction is essential in this usage scenario.

The benefit of flexible deployment of computing resources enabled Inter-Cloud Architecture (ICA) which appropriates multiple cloud service provider for better Quality of Service(QoS), reliability, cost effectiveness, and preventing vendor lock-in problem [29]. Massively parallel applications such as workflow management system such as Pegasus [17], parameter sweeps and bag-of-tasks [66] can be benefited both in terms of cost and reliability.

Naturally, since we can obtain computing resources from any cloud providers, we can diversify the geographical location of computing resources and increase the proximity between the server and client. Especially, in the field of content delivery network, where the latency between client and server is critical, diversifying locations can lead to better quality of service. Also, since, we are not dependent on a single cloud service providers, we can also increase the reliability, cost-effectiveness and can avoid vendor-lock in problem.

Inter-cloud architecture uses the Internet as a substrate network. However, the internet itself is a coordination of heterogeneous autonomous network. And Most of Infrastructures are outside the control of any given entity And computing resources on cloud are usually behind firewall and NAT router.

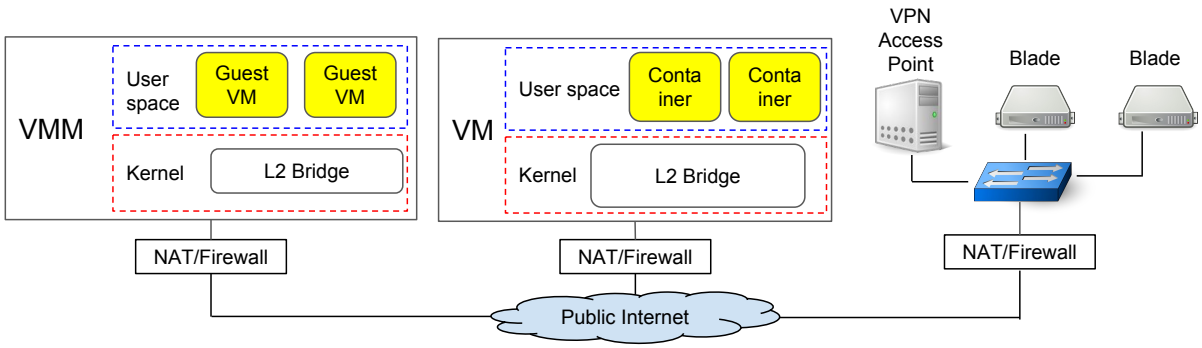

Figure 2-2. Inherent heterogenity of network environment for inter-cloud architecture.

So, it is difficult to merge into single LAN environment across ICA. Usually different private IP subnets are assigned in different cloud service provider. And not all providers support layer 2 network among the instances nor expose whole layer 2 network to the instances.

Moreover, computing resources may be provided in a different form as shown in Figure 2-2. Generally, it is provided in a form of guest VM from the virtual machine monitor. However, it is also now becoming popular to be provided in a form of container from a VM. However, in few cases, cloud services provide bare-metal machine with HW switches.

Overall, it is basically impractical to physically interconnect computing resources in ICA. So it is inevitable to use virtual network.

2.1.2 Nested Virtualization and Nested Virtual Network

Utilization of VIAS can be perfectly fit in nested virtual environment, which is recursively virtualizing virtual network. Since the rise of multi-tenant clouds, the approach of providing virtualizes computing resources such as virtual machines and containers has become key to an increasing body of distributed computing applications. As Infrastructure-as-a-Service (IaaS) becomes pervasive, tenants have the ability to deploy distributed computing services on demand across multiple private and public clouds. Furthermore, tenants are able to deploy their own virtual infrastructure on provider resources, using techniques that can be broadly characterized as "nested virtualizatio", not only as a means to improve utilization of resources, but also to enhance performance by co-locating services. To effectively use such

deeply-virtualized, distributed cloud environments, seamless networking across (nested) VM instances is key. Today, computing resources are readily available through multiple cloud service APIs and portals, and can be customized and/or instantiated from a template image in a matter of minutes. Cloud providers expose such virtual resources with a set of instance types, with discrete combinations of CPU, memory and storage capacities. However, the resource offerings may not match the requirements from tenant workloads, and may impact resource utilization, motivating several approaches that leverage nested virtualization [75, 97, 8].

Nested virtualization allows tenants to provision "inner" VM instances within the "outer" VM provisioned by the cloud service. As such, tenants can choose their own "inner" VMM and cloud stack (which may be different from the provider's) to run within the provisioned instance, allowing for increased flexibility – e.g. by slicing a generic large "outer" instance as several "inner" instances with tenant-configured sizes. Furthermore, nested virtualization allows tenants to configure once, and deploy their entire environment across different cloud service providers.

Such features are important because, while libraries such as libcloud [5] in principle allow tenants to manage multiple providers, in practice the problem of heterogeneity of services cannot be resolved just by using such libraries. Issues include that different providers may use different VMMs with different image formats, different block storage abstractions, and different network capabilities.

While nested virtualization adds a layer of indirection that unlocks benefits in flexibility and fine-grain control of resource allocations, it raises challenges with respect to performance and networking. One incarnation of nested virtualization that addresses the performance issue within an instance is to deploy light-weight containers (e.g. LXC, Docker) as the "inner" instances on top of a provider's hypervisor.

While tenants can use cloud APIs and Web interfaces within a provider to configure the network for their cloud instances (e.g. boto or gcloud), there is no inter-operability across providers.

In its most general form, nested virtualization allows the capability of virtual machines to themselves host other virtual machines, in a recursive fashion. It builds upon the ability of "classic" VM monitors/hypervisors to expose the abstraction of a virtual machine at the Instruction Set Architecture (ISA) layer, and can be supported if the underlying ISA follows the guidelines set forth in seminal virtualization work [71]. Initially applied in partitionable mainframe systems, nested virtualization became feasible in commodity server systems with the advent of virtualization extensions for the x86 architecture [90], and has been motivated by use cases in cloud computing [8].

While the most general form of nested virtualization allows a "classic" hypervisor to be instantiated within a virtual machine and supports completely unmodified software to run within VMs (e.g. instantiating KVM within VMware), the nested approach can also be applicable in other configurations, such as nested para-virtualized hypervisors (e.g. Xen on Xen [94]), trading off potential performance benefits with the additional requirement of software modifications in the kernel and/or applications. In particular, a form of nested virtualization that is appealing for deployment of software services in IaaS cloud computing platforms is to use the hypervisor of the cloud provider (e.g. Xen in Amazon EC2, KVM in Google compute engine, Hyper-V in Microsoft Azure) to deploy O/S containers (e.g. Linux LXC/Docker). This is the approach taken in Kangaroo [75]; the key advantage of this approach is performance, because containers are light-weight. The requirement to run software within containers poses a constraint, as nested instances need to use the same O/S kernel, but it is acceptable in many applications, as the adoption of container technologies continues to increase.

Different approaches to nested virtualization, as described above, yield different models of how virtual CPUs, memory and storage are allocated and managed. For instance, classic VMs expose virtual CPUs, physical memory, and block storage devices, while containers expose processes, virtual memory, and file systems at the O/S layer. Nonetheless, in general, nested virtualized systems typically expose a similar interface to the networking subsystem across multiple platforms. The lower-layer has Ethernet virtual network interface(s) provided and

managed by a cloud provider's "outer" hypervisor; these are multiplexed among Ethernet virtual network interfaces managed/exposed by the tenant's "inner" virtualization stack.

The increasing variety of cloud providers and the added flexibility of an extra layer of indirection from nested virtualization are welcome news for tenants interesting in deploying multi-cloud virtual infrastructures that seek to achieve high availability and performance while minimizing cost, and avoiding vendor lock-in.

Networking, however, becomes harder – it requires the coordinated configuration of virtual network interfaces, switches and routers across multiple instances. Furthermore, each cloud infrastructure has its own networking model and stack, and may be subject to connectivity constraints such as Network Address Translation (NAT), in particular in private clouds.

This leads to management complexity that is greatly compounded when tenants distribute their applications across clouds and expect elastic provisioning and management operations, such as VM migration, to work seamlessly.

Throughout this dissertation, we use the term "nested VMs" (or "nVMs" in short) to refer broadly to any nested virtualization technique that exposes a layer-2 virtual network interface to each instance. In the evaluation, we focus both on O/S containers as nVMs and physical hosts as well. The layer 2 virtual/physical networking techniques in VIAS generally apply to any nested virtualized system that uses layer-2 Ethernet virtual interfaces.

2.1.3 Networking in Nested Containers

To the best of my knowledge, all VMMs used in nested virtualization support the TUN/TAP (or equivalent) virtual network interface device for nested VMs. In particular, in Linux Containers (LXC) one or more TUN/TAP devices are created for each container. By default, LXC also creates a Linux bridge on the host, and connects all virtual network interfaces of the nested VMs to this bridge. In this way, all the nested VMs attached to this bridge reside on the same layer-2 Ethernet segment. Because multiple nested instances share a single interface of the lower-layer hypervisor, it is necessary to assign and manage addresses, and multiplex access to the single interface using address translation. To this end, LXC also

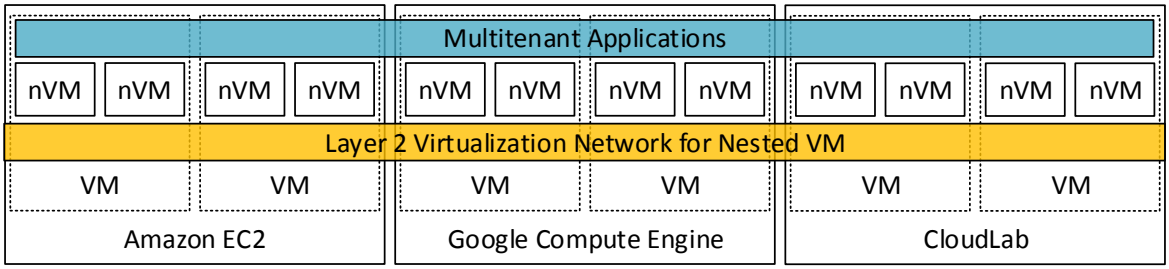

Figure 2-3. Layer-2 network virtualization for nested VMs across three cloud providers.

creates Network Address Translation (NAT) rules in the host machine, using Linux iptables. To support automatic address assignment, LXC runs a lightweight DHCP server on the host machine. Thus, upon instantiation, each nested VM gets assigned a random IP address within a predetermined subnet range of (private) IP addresses. While not all nested virtualization technologies automate the process of providing a layer 2 network environment behind a virtual NAT, this behavior can be programmed using existing Linux devices and toolsets – a Linux bridge, iptables and dnsmasq.

2.2 Network Virtualization

The purpose of VIAS is to provide a complete layer 2 abstraction to any host deployed across heterogeneous cloud services, without significant performance degradation. Especially, the goal is to allow distributed applications (possibly multi-tenant) within nVMs to seamlessly communicate as illustrated in Figure 2-3, as if they were connected to the same layer-2 segment – even though they are distributed across independently-managed providers.

In general, there are two approaches to tackle this issue. The first relies on tenants deploying their own overlay network – which has the key advantage of not requiring any support from the underlying infrastructure. The other approach is to exploit SDN and/or Network Function Virtualization (NFV) services provided by the cloud provider – which is not inter-operable across providers.

2.2.1 Overlay Networks

In overlay networks and VPNs (Virtual Private Network), the entire header and payload of a virtual network packet are encapsulated (and possibly encrypted and authenticated) by another layer to transfer the packet over public network links. Tunneling techniques such as L2TP, GRE or MPLS take advantage of encapsulation, which prepends an additional network header to the same or different OSI layer of the packet [80]. Tunnels can be built as stand-alone point-to-point links, or organized to form a topology, such as a mesh or structured P2P, that can be used for scalable overlay routing [95, 46, 47]. Overlay and tunneling techniques benefit from the flexibility of using encapsulation at the endpoints – which does not require changes to the infrastructure – but suffer from performance degradation. The additional encapsulation header is a source of overhead, limiting the effective Maximum Transmission Unit (MTU) size. Furthermore, overlay processing adds computation overhead of dealing with encapsulation – possibly at the user level, as typical overlay networks are implemented as user-level processes.

2.2.2 Software Defined Networking

Software Defined Networking initially emerged from the necessity of testing experimental protocols to overcome the difficulties of deploying new protocols on legacy hardware switches and routers [57]. Subsequently, the need for efficiently supporting large multi-tenant enterprise data centers, such as public and private clouds, has motivated the adoption of SDN techniques as an approach complementary to NFV (Network Function Virtualization) for data center network virtualization. Virtualization in cloud computing impacts the network performance because of its inherent sharing of processor resources. This can lead to negative impact on network performance and stability, such as exceptionally long delays and degraded throughput [91]. SDN and NFV techniques can mitigate performance degradation by migrating network virtualization processing overheads to network devices, and possibly lead to substantial reductions in operating expense for cloud and network service providers [38]. For example, VMware [51] supports network virtualization through both logically and physically deployed

SDN nodes, providing a Network Virtualization Platform (NVP) within a multi-tenant data center. NVP leverages software switches on VMware hypervisors at each server endpoint. However, cloud providers often constrain network communications available across instances; furthermore, SDN integration and layer-2 messaging outside a domain is not possible, hindering the ability for tenants to deploy their virtual networks across providers.

Typically, tunneling and SDN approaches are used in different network virtualization contexts – such as across and within a data center. VIAS seeks to integrates these two approaches for a flexible, cross-cloud overlay network virtualization technique that mitigates performance degradation by selectively applying SDN to establish intra-cloud fast bypass flows.

2.2.3 NAT Traversal

As described above, one challenge with nested virtualization is the need to multiplex the host network across multiple nested VM instances. Consider, for instance, a tenant using a VM provisioned by a cloud provider (cVM1), and then instantiating nested instances (nVM1...nVMi) within cVM1. Each nVM is considered by the tenant, and hence has a virtual address that is private. While the nVMi instances can communicate within cVM1 through a virtual bridge or switch, in order to communicate across multiple instances (e.g. to nVMs hosted in other machines in the same provider, or on a different provider) it becomes necessary to map and translate addresses. It is also possible that the VMs provisioned by a service themselves have private IP addresses that are translated by a NAT device, which is common in private clouds.

Hence, in the absence of the ability to provision public addresses to nested VM instances (which is currently not offered by cloud providers), network virtualization techniques must deal with multiplexing and translation of nested/host addresses. This can, in principle, be accomplished through careful crafting of network rules by the tenant. However, this becomes complex and error-prone as the network increases in size.

VIAS leverages the ICE [92] and STUN [79] protocols integrated in IPOP [47] to support dynamic NAT traversal, allowing nodes to self-organize peer-to-peer links on demand. To

illustrate, assume two peers A and B, both behind distinct NATs, want to establish a direct P2P tunnel. Initially, A does not know the outermost (public) transport address of itself, and nor does it know B's outermost address. Conversely, B does not know A's. At first, each peer queries a STUN server to discover this transport address. At each site, independently, the NAT binds one of the available transport port numbers, along with its IP address, to each peer's private address and port number; furthermore, the STUN server replies with outermost transport to each of the peers. At this point, both peers know their outermost public address; to establish a tunnel, those addresses need to be exchanged. For this exchange, VIAS leverages IPOP's use of the eXtensible Messaging and Presence Protocol (XMPP), using an XMPP server in the public network as an intermediary. After the successful exchange, each peer can send packets with the destination transport address of remote peer's outermost address. For instance, if A sends a packet to B's outermost NAT transport address, the message is received by B's NAT, which translates the outer transport address to an inner transport address, then delivering the packet within the private network to reach the final destination B.

2.3 VIAS Overview

This section overviews a novel system – VIAS (VIrtualization Acceleration using SDN) – that delivers the flexibility of overlays for inter-cloud virtual private networking, while transparently applying SDN techniques (available in existing OpenFlow hardware or software switches) to selectively bypass overlay tunneling and achieve near-native performance within a provider. Figure 2-4 shows the architectural illustration of VIAS. VIAS, in large, consists of two modules, IPOP and SDN controller. IPOP handles overlay network related features, creation or termination of P2P link, while SDN controllers control datapath of OpenFlow devices. In addition, while the SDN controller takes responsibility of programming OpenFlow switches to virtualize TCP/UDP streams, IPOP handles features that SDN is unable to support, such as encapsulation, encryption and authentication. Architecturally, VIAS is unique in how it integrates SDN and overlay controllers in a distributed fashion to coordinate the management of virtual network links. The approach is self-organizing, whereby overlay nodes can detect

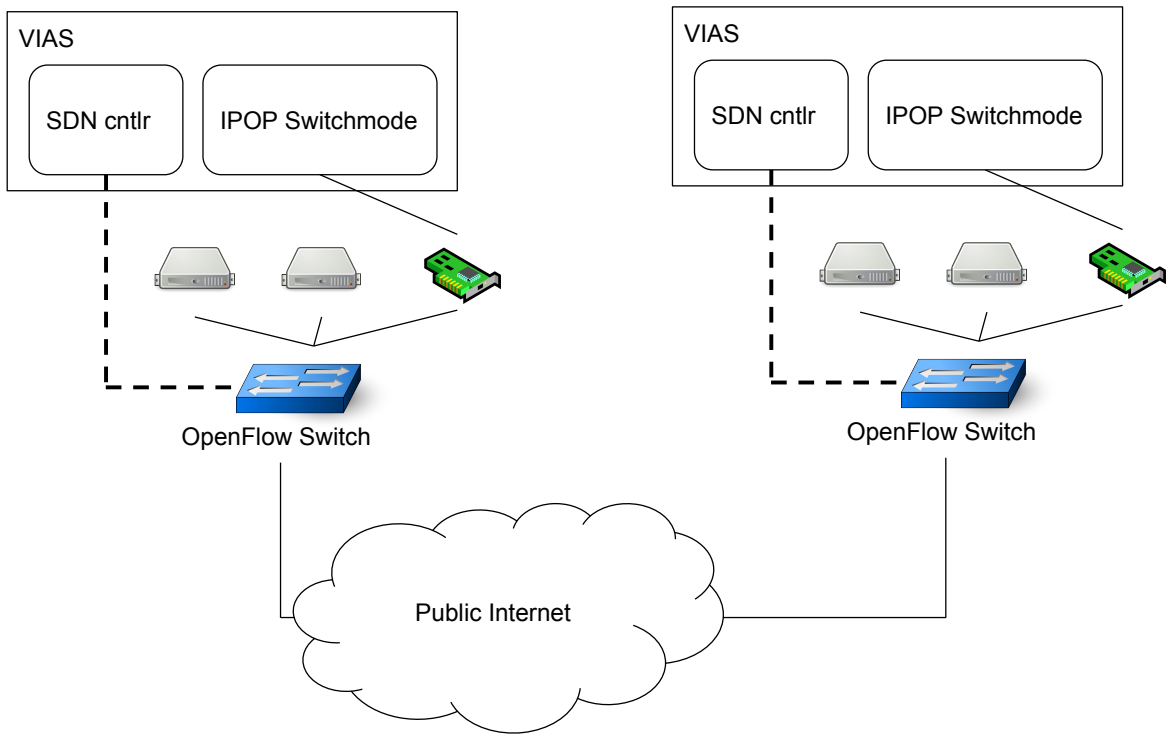

Figure 2-4. Conceptual illustration of VIAS

that peer endpoints are in the same network and program a bypass path between OpenFlow switches.

While generally applicable, VIAS targets tenants who use nested VMs/containers across cloud providers, supporting seamless communication among their nested resources without requiring any support, coordination, or standard interfaces across provider. VIAS has been implemented as an extension to an existing virtual network overlay platform, integrating OpenFlow controller support with distributed overlay controllers.

Prototype of VIAS has been deployed and tested in realistic cloud environments, using an implementation based on IPOP, the RYU SDN framework, Open vSwitch, and LXC containers across various cloud environment including Amazon, Google compute engine and CloudLab [16].

Network Address Translation (NAT) is heavily used, not only to deal with shortage of IPv4 addresses, but also in data center network to provide isolation. NAT is also essential in

nested virtual network environments, which separates guest VM/container network environment from host network environment. NAT inherently blocks unsolicited traffic from outside, accepting only traffic that is intended to receive. NAT dynamically translates from public IP and transport port number pair to private one and vice versa. NAT behavior has not been implemented in SDNs in previous work, but is important for cloud-provisioned inter-cloud overlays. This dissertation elaborates on design and implementation of a SDN NAT.

VIAS supports the general case (scalable, dynamic virtual private networking across providers possibly constrained by NATs) by employing overlay networking as the substrate for virtualization, and optimizes for a common case (communication among nodes within the same provider) by means of a novel performance enhancement technique of automatically detecting and programming fast bypath links using SDN APIs. Such SDN-programmed VIAS bypass removes the necessity of packet encapsulation and delivers virtualization performance near wire link speed. VIAS detects traffic-intensive TCP/UDP flows inside the overlay-encapsulated data traffic, and automatically switches over to the SDN fabric whenever such path can be programmed – such as when endpoints are within the same cloud provider. SDN switches on end nodes translate from outer/inner to inner/outer address space, which eliminates overlay headers, overlay processing, and user/kernel context switch.

2.4 VIAS Architecture

The key design requirements in VIAS are: 1) to expose a layer-2 abstraction to unmodified applications, 2) to operate as a user-level application that can be deployed in any existing cloud instance, thus not requiring development of kernel modules, 3) to support private, encrypted tunnels in inter-cloud communications, and 4) to avoid the overhead of encapsulation and kernel/user crossings for TCP/UDP flows within a cloud provider. The VIAS architecture addresses these requirements by: fully supporting layer-2 tunneling through TLS/DTLS P2P overlay links as a baseline functionality implemented in user-level overlay software, and by automatically detecting and programming SDN switches to bypass TCP/UDP flows.

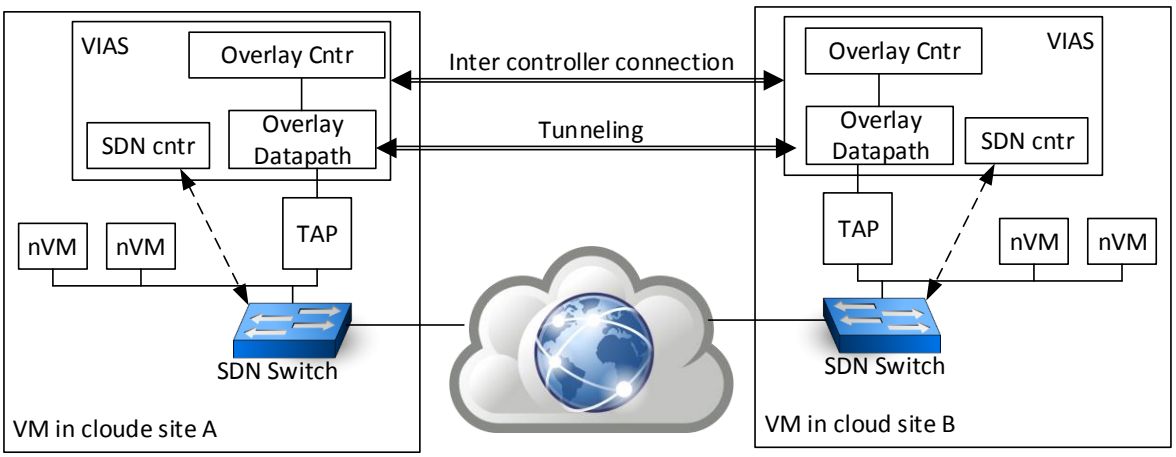

Figure 2-5. VIAS overall architecture.

There are important reasons why VIAS bypasses TCP/UDP flows via SDN switches, but carries other virtual network traffic (e.g. ARP, ICMP) in tunnels. First, TCP/UDP are the transports used by the majority of cloud applications and the common case that needs to made fast. Second, cloud providers typically allow instances to communicate using TCP/UDP, but block layer-2 protocols (such as ARP); since it is assumed that SDN switches are available only at the endpoint at the minimum (e.g. software switches such as Open vSwitch in an instance), VIAS is not necessarily able to program the cloud provider's entire switch fabric.

With existing OpenFlow standard APIs and existing functionality in cloud platforms, bypass flows can be programmed by coordinated mapping/translation of TCP/UDP flow endpoints at SDN switches connected to VIAS virtual network interfaces. To accomplish this, the VIAS architecture is structured as illustrated in Figure 2-5. It comprises of three main modules. VIAS main modules are user-level applications that are responsible for managing bypass flows (SDN cntr), setup and configuration of overlay tunnels (Overlay cntr), and encryption, encapsulation, tunneling and NAT traversal (Overlay datapath). The VIAS overlay datapath binds to an SDN switch port through a virtual network interface (tap), and the SDN controller module programs the switch. The SDN switches are commodity software/hardware

devices programmed using OpenFlow APIs; they are also referred to as vias-nat-switch in this dissertation. Detailed functionality of each modules follows as below.

2.4.1 SDN Controller

This module acts as a controller for the SDN switches. In essence, it allows OpenFlow switches to retain conventional features (such as MAC learning) but, in addition, implements the distinctive features of VIAS: the abstraction of a layer-2 virtual network, and isolation of the virtual network from the host network. The VIAS SDN controller programs gateway and NAT functionalities on the SDN switch, such that virtual network endpoints can have a different subnet and private address range from the host network's. Those features are programmed using standard OpenFlow APIs, without any modifications to the switch, allowing the use of commodity hardware and software implementations of SDN switches. The main requirement is that the VIAS module is allowed to issue OpenFlow API calls to the switches. For brevity, we will refer to the SDN switches with these capabilities as vias-nat-switch.

The vias-nat-switch also implements VIAS overlay bypass to achieve high throughput for intra-cloud TCP/UDP flows. These flow bypass rules are programmed (also using OpenFlow APIs) to have higher priority than that of other flow rules. We give more details on the implementation of this vias-nat-switch and overlay bypass in Subsections 2.5.1 and 2.5.2

2.4.2 Overlay Datapath

Packets captured by the tap virtual network interface are handled by this module. This module runs as a user-space process that reads/writes from the tap device, and executes all low-level packet processing functions, such as prepending or translating network headers, as well as encryption and authentication. VIAS leverages the IPOP [47] overlay stack for this module. It reads the destination address to look up an overlay destination, prepends IPOP and UDP headers, then injects the packet to the WAN interface.

While the creation and termination of P2P overlay links are managed by the overlay controller, the metadata associated with each P2P overlay link (such as peer UID, IPv4 or MAC addresses) are used by the overlay datapath module to make forwarding decisions.

Likewise, the necessary attributes of these headers (such as tunneling identifier and mapped IP and port numbers) are dynamically assigned by the overlay controller and programmed into the overlay datapath module. After the UDP header prepending, packets are ready to traverse through tunnels across the public Internet. The design and implementation of this module is elaborated in Subsection 2.5.2.

2.4.3 Overlay Controller

The creation and termination of P2P links among overlay nodes, and the topology of the overlay network, are managed by this module. The VIAS overlay controller extends the IPOP overlay controller, which currently supports three types of topologies. One is an unstructured social network graph topology, where P2P links are created based on social links. A second topology is an all-to-all graph connecting all nodes, while the third topology is based on the Chord [85] structured P2P system. VIAS can use the all-to-all topology for small networks (tens of hosts), or the structured topology to scale to larger nodes. In the structured topology, each node identifier is based on SHA-1 of its virtual IP address, and identifier-based structured routing is performed when there is no direct P2P link connecting nodes. Structured topology policies, such as the number of successors or chord links, are configurable in the controller.

2.4.4 Overlay Link and Bypass Flow Management

The general approach taken by VIAS for the management (creation, monitoring, and tear-down) of links is as follows. First, overlay links are created by the overlay controller. Links may be created to enforce a topology invariant (e.g. left/right neighbors and chords in structured P2P), or on-demand based on traffic initiated by a peer. Second, for active links, their state is monitored (with respect to their traffic, and the transport addresses) by the overlay datapath module, and made available to the overlay controller. Building on overlay link monitoring mechanisms, the overlay controller defines policies for establishing a bypass TCP/UDP flow that takes into account: 1) the traffic between nodes, and 2) whether the two endpoints are within the same provider network, to initiate a link a request to create a bypass flow. The mechanisms to initiate a bypass flow are handled by the SDN controller module.

There are two instances where a bypass flow might be terminated: one is after VM migration, and another is if resources (available ports) are exhausted at the SDN switch. The policies to deal with bypass termination are left for future work, but mechanisms available to support these policies are monitoring of traffic per flow (which can be used to prioritize high-throughput traffic), and migration events (e.g. unsolicited ARP requests) to notify controllers to terminate a flow. In such events, the SDN controller can then clear out bypass rules, hence reverting all virtual network traffic to pass through overlay tunnels.

2.5 VIAS Imeplementation

In this section, details of a VIAS prototype implementation is provided, highlighting how each module is implemented and integrated. Moreover, we explain how the transition between overlay and SDN virtualization takes place in VIAS. In Subsection 2.5.1, it is elaborated on how the dynamic NAT feature offered by VIAS is programmed into the OpenFlow SDN switch. In Subsection 2.5.2, tunneling and overlay virtualization along with various tunneling modes are explained. Finally, in Subsection 2.5.3, it is described how VIAS detects flows and implements rules to bypass overlay virtualization by using SDN flow rules.

2.5.1 SDN Controller

As explained in Section 2.1.3, each nested VMs should be presented the full abstraction of a private layer 2 network, while still able to access the public Internet as if behind a NAT. To support this requirement, VIAS programs address translation rules using the OpenFlow controller, which makes Open vSwitch (or any other OpenFlow enabled devices) to work as a full-cone NAT router.

Through a configuration file, VIAS specifies a single switch port as its WAN interface (either by the physical port number of a switch or interface name); the remaining switch ports are set as LAN ports. The controller also implements a gateway. Both the subnet range and the gateway IP address are statically assigned through VIAS configuration, and VIAS divides the address space of nested VMs from the cloud providers' address space. When packets are sourced from the LAN address range and destined beyond the gateway (i.e. the

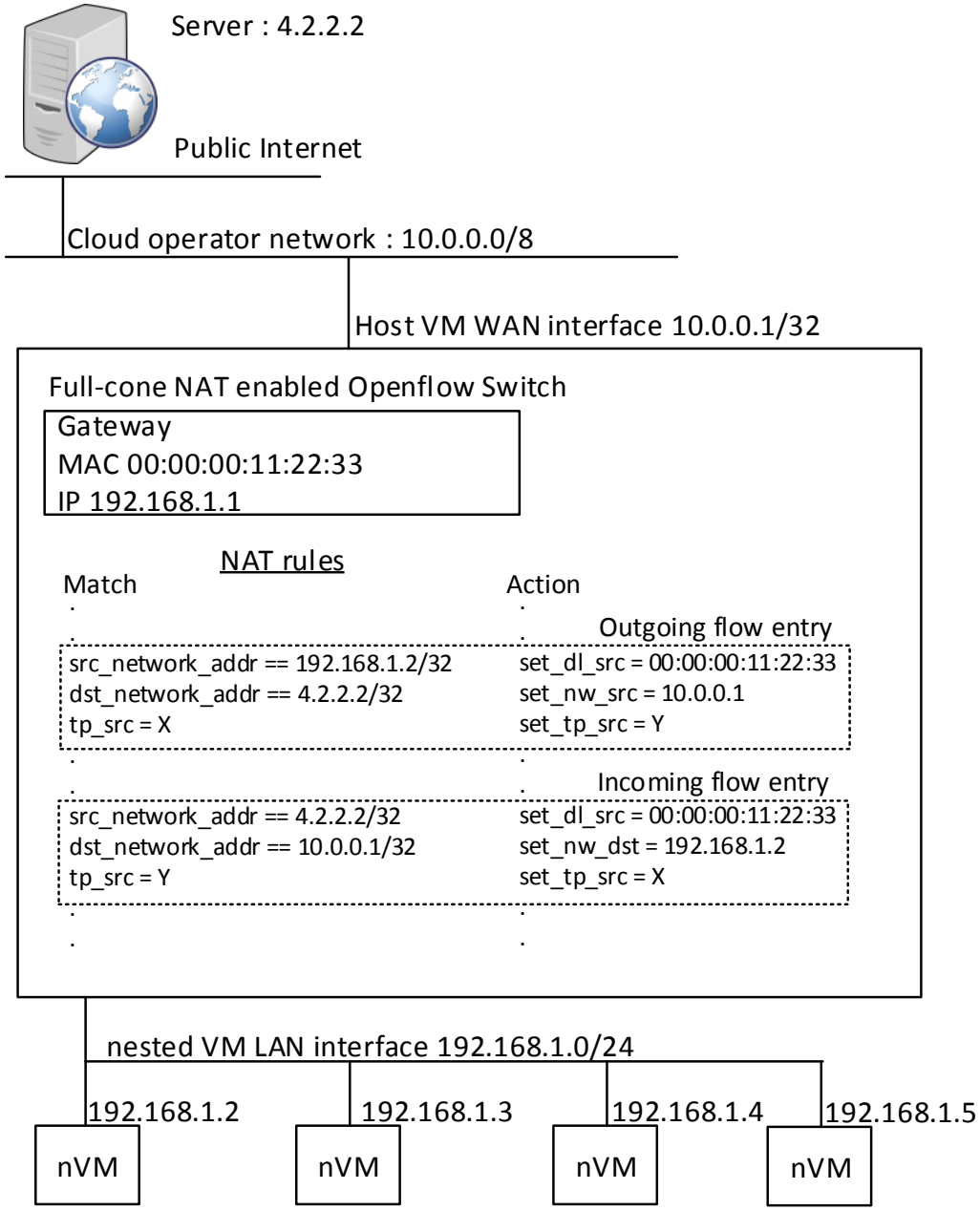

Figure 2-6. NAT implementation in VIAS using OpenFlow protocol.

destination address is out of the range of LAN subnet), the controller programs a flow rule in the OpenFlow switch to perform full-cone NAT mappings.

For example, as in Figure 2-6, the VIAS SDN switch is programmed using OpenFlow to support NAT translation, in addition to "upcalls" to the overlay datapath and overlay bypass rules. This allows nVMs to access public Internet nodes through a layer-3 NAT gateway, in addition to exposing a virtual layer-2 network across all nodes connected to the overlay. In the example, an IP packet sent by nVM 192.168.1.2 to public server 4.2.2.2 triggers the programming of NAT rules by the VIAS SDN controller. In detail, assume a nested nVM with IP address 192.168.1.2 tries to access a public server 4.2.2.2 through TCP with source port number X. Initially, the destination address of the very first packet does not match any existing flow rules in the switch. Note that this controller supports MAC address learning, such that it can learn and map each nVM MAC address to its respective port. Then, the metadata of this first packet is forwarded to the SDN controller through the OpenFlow API of ofp_packet_in. The SDN controller then checks the destination address; it can determine that the destination 4.2.2.2 is not in its LAN segment, but instead across the gateway. The SDN controller then randomly chooses port number Y from an available port number pool (which is also configurable). Finally, it makes two flow entries: one is an outgoing flow entry for streams from nested nVM to the public Internet, and the other is an incoming flow entry for the reverse stream. This can be done with series of ofp_match and ofp_actions.

Continuing in the example, the outgoing flow entry translates address from 192.168.1.2:X to 10.0.0.1:Y, replacing the source address of nested nVM to its WAN interface. The incoming flow entry translates destination address from 10.0.0.1:Y to 192.168.1.2:X, and injects the packets to the switch port of destination nested VM. The same technique can be applied to UDP streams.

Since the absence of ICMP echo identifier field in match of OpenFlow specification, ICMP protocol, which is widely used for echo request/reply (ping) message, cannot be NATted. One limitation of this SDN controller is that it cannot apply this NATting behavior to ICMP echo request/reply (ping) messages, a protocol widely used for testing reachability of network layer 3 devices. OpenFlow increases the type of ofp_match field from 10 of version 1.0 to 45 of version

1.5. However, the ICMP echo identifier field, which is widely used for NATting in conventional routers, is not incorporated in the OpenFlow specification. Therefore, it is impossible to apply above technique of ofp_match and ofp_action operation to ICMP messages. If future OpenFlow specifications include this field, we can apply the same approach by using the ICMP echo identifier instead of transport port number. To this end, ICMP messages are forwarded to the controller, and NATting is performed in the SDN controller. While this approach increases the latency of ICMP messages, the functional behavior of the protocol is unaltered. Since ICMP is not used for traffic-intensive communications, this performance degradation is acceptable.

In the VIAS prototype, the SDN controller has been built upon the open-source RYU [6] SDN framework. The code (except the framework) is approximately 800 lines of Python code. For full backward compatibility, this implementation is based on OpenFlow specification version 1.0.

2.5.2 Overlay Datapath and Overlay Controller

VIAS builds upon the IPOP codebase as a basis for the overlay datapath. In its current architecture, IPOP comprises of a packet processing/overlay datapath binary, and an overlay controller, which communicate through the TinCan API [47]. In essence, VIAS extends the IPOP overlay controller to support SDN bypass processing – these two modules are embodied in VIAS as explained in Section 2.4.

In IPOP, nodes create direct P2P overlay links to tunnel virtual network traffic using the ICE protocol [92]. In order to bootstrap these direct P2P overlay links, an XMPP server is employed to assist in exchanging messages containing candidate endpoint information of peers – including the outer- and inner-most transport pairs, if nodes are behind multiple NATs. During this process, IPOP's overlay datapath module opens UDP ports ("hole-punching") on NATs to create P2P tunnels. Subsequently, peers communicate using these assigned UDP transports. To capture packets within the overlay address range from the O/S network kernel, a virtual network interface device (tap) is used. After the packet is captured by the tap device, it is forwarded to the IPOP overlay datapath module. This module prepends an IPOP header

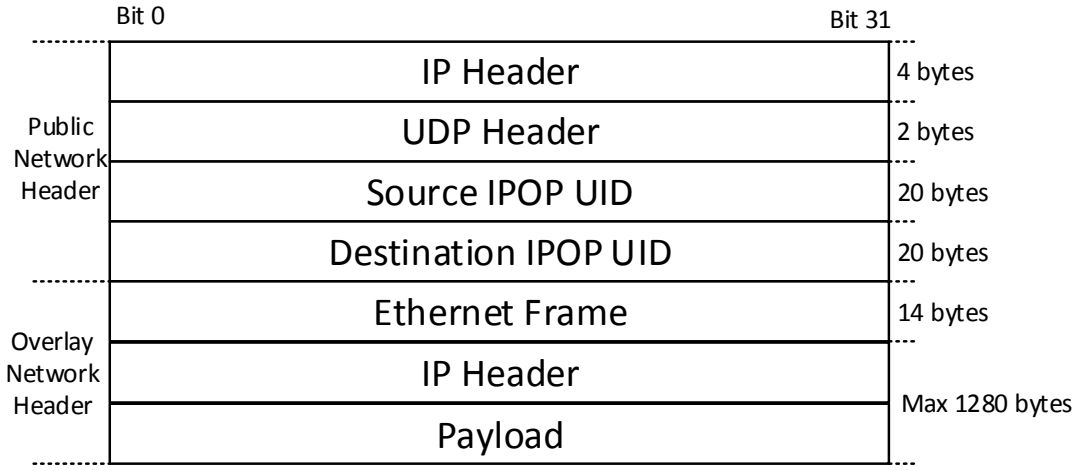

Figure 2-7. Packet encapsulation in IPOP.

to the packet; it is then encapsulated again by UDP header, and then send to the destination peer's UDP port, which is discovered through the ICE protocol and punched in the NAT by the remote peer. Each IPOP overlay node is assigned a 20-Byte unique identifier (UID) that is used in overlay routing. The IPOP overlay header consists of two fields, which are the source and destination UIDs of overlay sender and receiver. The outer UDP and IP header are placed before the IPOP headers, resulting in the overall packet structure as shown in Figure 2-7.

IPOP maps various network address identifiers (MAC, IPv4 and IPv6 addresses) to P2P links, and currently supports both layer 2 and layer 3 virtual networks. VIAS uses the switch (layer 2) mode of operation. IPOP keeps four separate hash tables: p2plink_table, ipv4_table, ipv6_table, and mac_table. The key of p2plink_table corresponds to a UID of the remote IPOP peer, while the value corresponds to the link object. The ipv4_table, ipv6_table, mac_table map IPv4, IPv6 and MAC address respectively to IPOP overlay node UIDs, which is the key of p2plink_table.

In VIAS, each IPOP overlay link is considered as an OSI layer 2 tunnel. Each IPOP link is mapped such that multiple MAC addresses are bound to a link – which is akin to a layer 2 switch's MAC addresses bound to a port. The layer-2 overlay implements learning of MAC

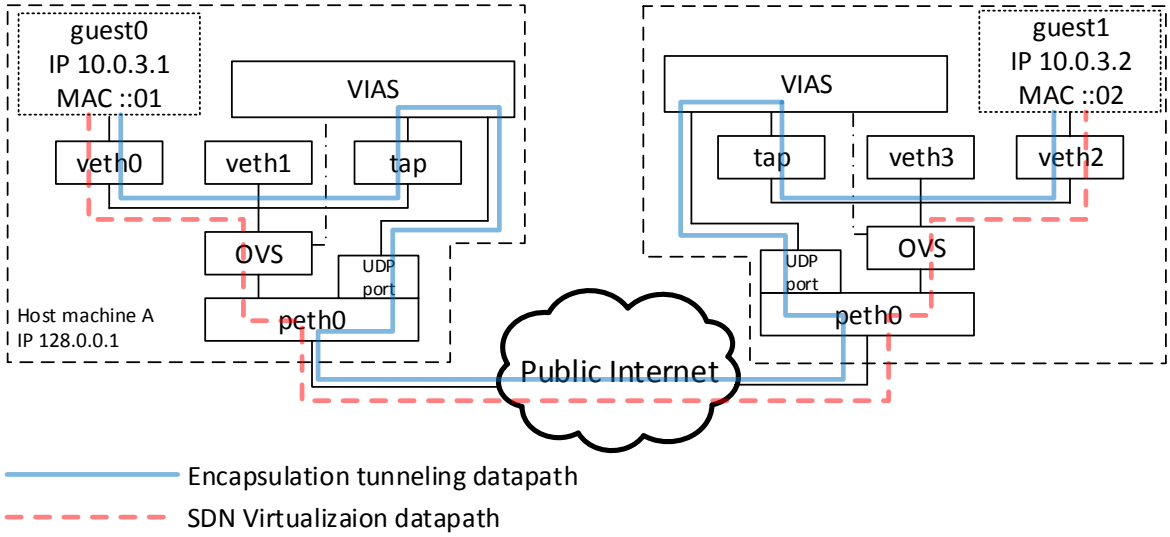

Figure 2-8. Packet datapath alternatives in VIAS.

addresses by handling ARP request/reply messages in its LAN. IPOP checks the destination MAC address of the frame and injects the frame to the corresponding link. To bind the virtual network to VMs, IPOP uses virtual devices, including Linux bridge and Open vSwitch [70]. The tap interface is attached to this binding device and works as a bridge to the remote LAN virtual network.

For example, consider the usage scenario with Figure 2-8. The illustration shows two hosts running VIAS-enhanced IPOP. Each host contains multiple nested VMs (containers) with virtual network interfaces (veth#) attached to the Open vSwitch device. These guests are in the same layer 2 network, and are NATted by the host network stack to the physical interface (peth0). Initially, packets flow through the encapsulation tunneling datapath (solid line), through the overlay datapath module. Once a bypass flow is installed by the SDN controllers on both endpoints, the faster SDN virtualization datapath (dashed line) is used. When guest0 (veth0) attempts to send IP packets to guest1 (veth2), it first broadcasts an ARP request message. The VIAS-IPOP overlay datapath module picks this message through the tap device, and handles it as follows. First, the ARP message is encapsulated (Figure 2-7) and forwarded

to all overlay links. At this stage, different overlay multicast approaches can be implemented by the overlay controller, depending on the overlay's topology. All the overlay nodes receiving the overlay-broadcast packet (e.g. the right-hand side of Figure 2-8) decapsulate the message and broadcast it to its L2 network (using the tap device). If there is no destination matching the ARP request message, the message is simply dropped. If the destination is in the network (e.g. veth2 in Figure 2-8), an ARP reply message is created by the guest, and the reply is sent back (with an unicast message) to the sender (e.g. guest0).

As part of this process, the overlay creates entries in mac_table binding the MAC address it learned to the corresponding overlay link. As part of this process, the overlay maps the MAC address it learned to the corresponding overlay link. All unicast MAC address frames captured by the overlay look up these mappings to determine along which overlay link to forward. VIAS mode can dynamically accommodate overlay topology changes by updating an MAC address and its overlay link bindings upon detecting an ARP frame. If we consider a usage scenario of VM migration from one host in a provider to another host in a different provider, the process can be automatically handled; there needs to be no network administrator involvement, since the ARP message from the migrated node itself incur updates the MAC-overlay link mapping on deployed overlay network.

2.5.3 Software-defined Overlay Bypass

As described in the previous section, the overlay virtualization process places several sources of overhead. At first, there is the transition overhead of context switch between network kernel to user space. It also requires multiple copy operations when it sends packet from network kernel to user space. Moreover, since there is the need for additional prepending of overlay headers, the MTU size is smaller than that of the physical network. This overhead is the price paid to create overlay virtual networks linking nVMs across multiple providers, because tunneling, NAT traversal and encryption is required for virtual private networking. However, this overhead can be mitigated for nVMs "within the same provider". To accomplish this, we bypass this encapsulation process on traffic-intensive TCP/UDP flows, as follows.

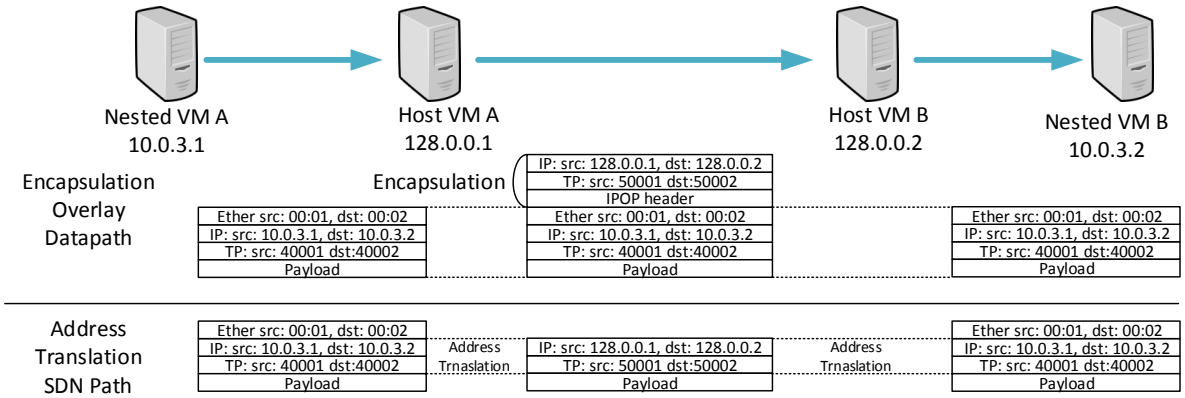

Figure 2-9. Virtualization comparison of datapath

VIAS detects a traffic-intensive stream by monitoring traffic on overlay links, and can determine if an overlay path is possible by inspecting the overlay endpoints of a link whether they are within the same network. When VIAS determines that a bypass link should be created, it first allocates and assigns an available TCP (or UDP) port number pair for the outer address space. Then it establishes a mapping of inner address range to the outer address range with its port number. In OpenFlow, this can be programmed with a single flow add API call with ofp_flow_mod along a set of ofp_match and ofp_action.

On matched packet on ofp_match, the VIAS controller performs actions OFPAT_SET_DL_SRC and OFPAT_SET_DL_DST to replace the MAC address of nested VM and Open vSwitch to physical Ethernet (peth) and gateway of outer address space. Then it performs actions OFPAT_SET_NW_SRC and OFPAT_SET_NW_DST to replace inner IP addresses to outer address space. Finally, actions OFPAT_SET_TP_SRC and OFPAT_SET_TP_DST replace the inner transport number to outer transport number. The incoming traffic takes the same steps, but changes from public address space to private address space.

To illustrate this behavior, consider the example of a TCP stream depicted in Figure 2-9 alongside the flow rule example of Table 2-1. The example uses the private address space 10.0.3.0/24 for the inner address space for nested VMs, and public network addresses 128.0.0.0/24 (note that it is also possible to use private addresses for the outer address space,

Table 2-1. Stream bypass rule example

		Match	Action
Local host	Outbound	nw_src=10.0.3.1, nw_dst=10.0.3.2, tp_src=40001, tp_dst=40002	set_nw_src=128.0.0.1, set_nw_dst=128.0.0.2, set_tp_src=50001, set_tp_dst=50002
	Inbound	nw_src=128.0.0.2, nw_dst=128.0.0.1, tp_src=50002, tp_dst=50001	set_nw_src=10.0.3.2, set_nw_dst=10.0.3.1, set_tp_src=40002, set_tp_dst=40001
Remote host	Outbound	nw_src=10.0.3.2, nw_dst=10.0.3.1, tp_src=40002, tp_dst=40001	set_nw_src=128.0.0.2, set_nw_dst=128.0.0.1, set_tp_src=50002, set_tp_dst=50001
	Inbound	nw_src=128.0.0.1, nw_dst=128.0.0.2, tp_src=50001, tp_dst=50002	set_nw_src=10.0.3.1, set_nw_dst=10.0.3.2, set_tp_src=40001, set_tp_dst=40002

since VIAS supports NAT traversal). Initially, packets stream through encapsulation overlay datapath, VIAS prepending headers every packet. At certain threshold of traffic, VIAS triggers SDN bypass. First it extracts metadata of the stream, including the source/destination IP address (10.0.3.X) and port number (4000X). Then VIAS allocates and assigns an available port number (5000Y) to use as outer transport address. Next, VIAS programs inbound OpenFlow rules on the Open vSwitch SDN switch in host VM A. These inbound rules translate outer transport (IP address and port number) to inner transport address. When VIAS programs an OpenFlow switch, it ensures the stream bypass rules have higher priority over other flow rules, so that the packet is not applied to other flow rules. Soon after, VIAS sends an inter-controller RPC message through its P2P overlay link to the peer overlay node, passing along JSON-formatted metadata of the stream as in Listing 2.1. This metadata sent through the inter-controller RPC API contains information such as outer transport address, inner transport address and transport type.

Upon receiving the message, the VIAS controller at host B programs inbound and outbound rules to its Open vSwitch SDN switch. The VIAS controller at host B then sends

Listing 2.1. Example of stream metadata exchanged by VIAS controllers to coordinate bypass flow rules.

```
{
  "protocol" : "TCP",
  "local_host_ipv4" : "128.0.0.3",
  "remote_host_ipv4" : "128.0.0.13",
  "src_random_port" : 50001,
  "dst_random_port" : 50002,
  "src_ipv4" : "192.168.4.3",
  "dst_ipv4" : "192.168.4.13",
  "src_port" : 40001
  "dst_port" : 40002
}
```

an acknowledgement RPC to host A through the overlay link, so that it makes sure that the outbound rule in local host is programmed only after all the other rules programmed. This ordering of events needs to be enforced to avoid packet loss during the setup of bypass rules. If we set local outbound rule before the other flow rules, the stream will have packets silently discarded by the SDN switch because of no matching rules. Finally, right after the final outbound rules are programmed in OpenFlow switches, the stream bypasses encapsulation and transfers packets through SDN switches on both endpoints.

This approach can be seen as an approach akin to NATs, as it provides the ability to map and translate addresses. However, unlike the conventional use of NAT, where each individual NAT is independently controlled, this scheme orchestrates the programming of mappings across controllers on both peer endpoints. VIAS essentially uses overlay link as control channels for coordination among the two peer SDN controllers to establish NAT mappings simultaneously across endpoints, allowing both nodes behind NATs to have a direct SDN flow that bypasses the overlay.

2.6 VIAS Evaluation

In this section, we evaluate VIAS from three different perspectives. Firstly, in addition to providing overlay networking and bypass paths, VIAS acts as an OpenFlow-programmed SDN bridge and NAT for nested VMs. Typically, nested VMs create virtual network interfaces bound

Table 2-2. ICMP and TCP performance comparison between Linux NAT and NAT featured openvswitch of VIAS

	Test Case		ICMP	TCP
Host VM	S1	Native	0.487 ms [±0.104]	5.24 Gbps [±0.421]
		Open vSwitch	7.76 ms [±0.790]	4.87 Gbps [±0.577]
		Percent change	1493 %	- 7.06 %
	S2	native	41.7 ms [±0.269]	547 Mbps [±10.4]
		Open vSwitch	49.2 ms [±1.35]	527 Mbps [±48.9]
		Percent change	17.9 %	-3.80
Guest VM	S1	Native	0.569 ms [±0.0611]	4.09 Gbps [±0.406]
		Open vSwitch	7.76 ms [±0.686]	3.86 Gbps [±0.518]
		Percent change	1264 %	-5.62 %
	S2	native	41.7 ms [±0.514]	411 Mbps [±85.6]
		Open vSwitch	49.3 ms [±0.978]	398 Mbps [±83.1]
		Percent change	18.2 %	-3.16 %

Standard deviation is shown in square brackets

to a host Linux bridge and NAT behavior is implemented using iptables. To evaluate the VIAS SDN bridge/NAT, we compare the performance of Open vSwitch-based VIAS to the native Linux bridge/NAT implementation. The second evaluation considers the throughput delivered by VIAS for TCP streams between nested VMs within and across cloud providers. Thirdly, we use an application-layer benchmark (Redis) to evaluate end-to-end VIAS performance. For all experiments, Ubuntu Linux 14.04 LTS hosts are provisioned from clouds. Software SDN switches (Open vSwitch version 2.0.2), and LXC containers inside the host are installed and configured to create nested instances. VIAS is implemented as extensions to IPOP 15.01 using RYU as a framework for OpenFlow handling.

2.6.1 VIAS Open vSwitch bridge/NAT

As pointed out in Section 2.5.1, OpenFlow is not capable to program flow rules to handle NATting of ICMP echo request/reply. Consequently, all ICMP echo request/reply packets are forwarded by the SDN switch to the VIAS controller: the controller itself handles NAT for ICMP packets by making an entry in a local table for every outgoing ICMP echo request message, using the ICMP echo identifier field as a key to this table. Even though the VIAS controller runs in the same host as Open vSwitch, ICMP handling incurs overheads.

Open vSwitch NAT overhead is compared with the native Linux bridge/NAT implemented by iptables in Table 2-2. In the experiment, host VMs are deployed in the CloudLab [16] IG-DDC Cluster in Utah and are provisioned with 862MB of memory. 50 ping tests (ICMP echo) and 10 iperf throughput (TCP) tests are conducted and the arithmetic mean (and standard deviation) is reported, considering a client in CloudLab and two different servers. One server (S1) resides in the same CloudLab cluster. The other server (S2) is provisioned as an "m1.small" instance with 2GB memory in the Chameleon cloud [13] in Texas.

As the results show, a latency overhead of about 7 ms is incurred in ICMP NAT handling, irrespective of the network distance to the server. This is because the overhead mostly comes from the inter-process communication between Open vSwitch and the VIAS controller, and kernel/user context switch. This overhead is acceptable, as the ICMP echo message is typically used for network reachability checks and not for latency-sensitive applications. The TCP iperf test results shows that the bridge/NAT throughput degrades about 3-7% compared to native Linux iptables. As Open vSwitch developers argue that the TCP performance is equivalent to that of Linux bridge [70], the slight performance degradation observed is due to the NAT rules programmed by VIAS. The NAT rules in the native Linux case are set by iptables and executed in Linux network kernel, while the VIAS NAT is implemented by flow rules inside Open vSwitch.

2.6.2 Microbenchmarks

This experiment evaluates the performance of VIAS for virtual network communication among nested VM instances. To this end, VIAS is deployed on multiple cloud service platforms, including commercial and academic clouds. This allows the evaluation of functionality and performance for intra- and inter-cloud deployments across various geographical locations. It is demonstrated that the nested containers separated by multiple NATs across multiple clouds are successfully connected by a virtual layer-2 network, and evaluate the performance of ARP, TCP, and ICMP protocols.

The following test cases are considered. The first case (CC) uses two Xen VMs deployed on CloudLab [16]. Each VM is provisioned with 862MB memory on CloudLab IG-DDC. The second case (AA) uses two Amazon EC2 instances of type t2.medium in the same zone (Oregon). Although Amazon does not provide specifications of network throughput of t2.medium (only mentioning that its low to moderate), based on the link test in Figure 2-4, the performance levels are commensurate to an 1Gbps Ethernet link. The third case (GG) uses two Google compute engine instances of n1-standard-1 at US central zone. Host physical machine was provisioned on Intel(R) Xeon(R) CPU @ 2.30GHz and 3.7GB memory. In the fourth case (AA_dz), experiments are also conducted with t2.medium Amazon EC2 instances, but with VMs distributed across two availability zones (N. Virginia and Oregon) for comparison. Finally, the fifth case (AG_dz) considers two instances deployed across two different cloud service providers. One instance is on Amazon EC2 (t2.medium) at Oregon and the other instance is on Google compute engine (n1-standard-1) at US central.

Column *physical* is the latency between host VMs. The *overlay* column shows the latency between nested VMs. The traffic streams through overlay datapath from Linux bridge to tap device to P2P tunnel. The *VIAS* column shows the latency of overlay datapath with Open vSwitch and tap device. Note that, for the VIAS column, ARP and ICMP is forwarded to SDN controller, incurring additional latency.

2.6.2.1 ARP

The ARP latency is measured using iputils-arping and is shown on Table 2-3. The test is repeated 50 times, and the arithmetic mean and standard deviation are reported. The results for the AA, CC and GG case show that the overhead of ARP handling in overlay is less than 1.5 ms, while in VIAS, the latency overheads are in the range of 4-24ms, due to inter-process communication and SDN controller processing. Surprisingly, the results show an exceptionally long latency of ARP across host Xen VMs in one particular environment CC – the overlay and VIAS latencies are smaller than physical network. While it was not able to definitively determine the reason for this behavior, one observation is that the VIAS and the overlay

Table 2-3. ARP and ICMP latency comparison among conventional Linux implementation, overlay datapath and VIAS

Test case		Physical	Overlay	VIAS
ARP [ms]	CC	432 [±243]	1.47 [±0.130]	24.4 [±0.102]
	AA	0.693 [±0.0357]	1.42 [±0.0805]	4.00 [±0.120]
	GG	N/A	0.4207 [±0.126]	4.63 [±0.372]
	AA_dz	N/A	84.5 [±0.243]	N/A
	AG_dz	N/A	49.7 [±0.116]	N/A
ICMP [ms]	CC	0.954 [±0.121]	1.17 [±0.0825]	15.4 [±1.20]
	AA	0.559 [±0.132]	0.970 [±0.149]	5.33 [±0.197]
	GG	0.421 [±0.126]	0.697 [±0.0843]	6.22 [±0.409]
	AA_dz	84.6 [±0.400]	92.5 [±0.169]	N/A
	AG_dz	50.7 [±0.101]	50.2 [±0.379]	N/A

CC: cloudlab AA:Amazon GG:Google compute engine AG:Amazon-Google dz:different zone. Standard deviation is shown in square brackets

paths use UDP as the protocol, whereas arping on the host VMs uses the ARP protocol. It is possible that the CloudLab platform handles ARP and UDP in different ways – since ARP traffic is relatively infrequent, the effects of delay in overall network performance are typically not significant. The ARP measurements in the AA case show that physical latency is the lowest.

Naturally, geographically separated nested VMs (AA_dz and AG_dz) exhibit longer ARP latencies due to network distance; this is observed in the results summarized in the overlay column. Note that because the AA_dz and AG_dz instances are not in the same LAN segment, ARP traffic in the physical network is not supported, hence the physical column shows N/A. Furthermore, it was not able to evaluate ARP in GG case, because Google compute engine does not provide layer 2 abstraction among instances deployed in the same zone, and thus the ARP protocol does not work. Nonetheless, the results demonstrate that, regardless of providers blocking L2 traffic within or across clouds, VIAS can present a L2 virtual network to nested VMs.

Table 2-4. TCP performance comparison among physical, overlay datapath and SDN bypass virtualization scenario.

Test case		Average	Standard Deviation	Percent change
CC	Physical	932 Mbps	±9.66	0.0%
	Overlay	92.8 Mbps	±6.97	-90.0%
	VIAS	901 Mbps	±7.42	-3.33%
AA	Physical	933 Mbps	±53.4	0.0%
	Overlay	293.4 Mbps	±8.47	-68.6%
	VIAS	876 Mbps	±49.6	-6.11%
GG	Physical	1.99 Gbps	±0.00471	0.0%
	Overlay	223 Mbps	±14.7	-88.8%
	VIAS	1.99Gbps	±0.00316	-0.00%
AA_dz	Physical	140 Mbps	±24.3	0.0%
	Overlay	41.18 Mbps	±17.3	-70.6%
	VIAS	71.17 Mbps	±43.9	-38.0%
AG_dz	Physical	118 Mbps	±27.1	0.0%
	Overlay	44.9 Mbps	±11.9	-61.9%
	VIAS	162.2 Mbps	±27.9	+37.5%

2.6.2.2 ICMP

ICMP echo latency is measured using the Linux ping command. The test is repeated 50 times, and report the arithmetic mean and standard deviation of the latency. The trend is similar with the ARP latency. The latency overhead is of the order of a few milliseconds across all test cases. Since the overhead only concerns in local kernel/user boundary crossing and local socket interface, the nested VM latency is not a function of the physical network's latency, but rather a constant overhead at the endpoints.

2.6.2.3 TCP

In Table 2-4 and in Figure 2-10, the TCP throughputs with different configurations between nested VMs across different cloud services are summarized. The iperf tool is used to test the maximum TCP throughput; tests were repeated 10 times, and the arithmetic mean and standard deviation are shown. For the physical row, iperf was executed on the host VM, while for the overlay and VIAS row, iperf was executed on the nested VMs.

The results show that encapsulation in user space has its peak throughput at around a few of hundreds Mbps, regardless of link layer bandwidth. This overlay performance is a function of

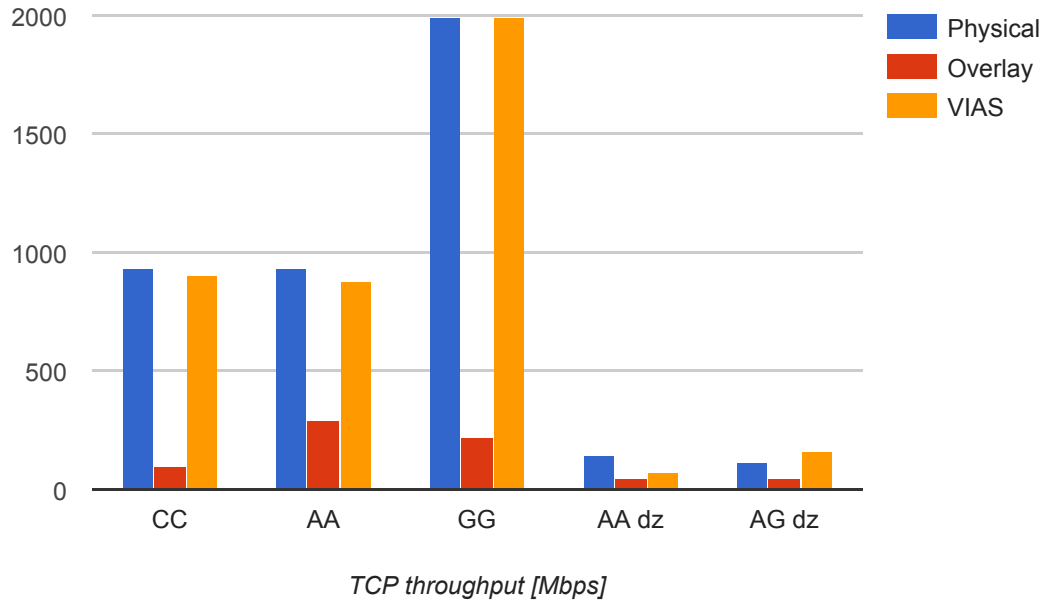

Figure 2-10. TCP performance comparison among physical, overlay datapath and VIAS virtualization scenario.

the host processor's performance, and overheads associated with packet handling and copying and O/S kernel-user context switch. In contrast, SDN virtualization achieves over 94% of the throughput of the physical network when endpoints are in the same data center (cases CC, AA and GG). Note that this experiment considers a software Open vSwitch that runs on the host VM, and does not use any assist from SDN hardware – though in principle VIAS can also program hardware-layer SDN switches, if available. Nonetheless, by eliminating O/S context switch and overlay packet handling through the VIAS SDN overlay bypass results in substantial performance improvements for network virtualization.

As described in section 2.5.3, VIAS detects traffic-intensive TCP streams at runtime, and then automatically inserts bypass rules in the SDN fabric. Prior to completion of flow rule, TCP streams traverse the overlay datapath. Thus, there is a latency involved in the coordinated programming of flow rules in the SDN switch of both peer endpoints. This latency is a function of the round trip time between host VMs: The SDN bypass setup latencies are

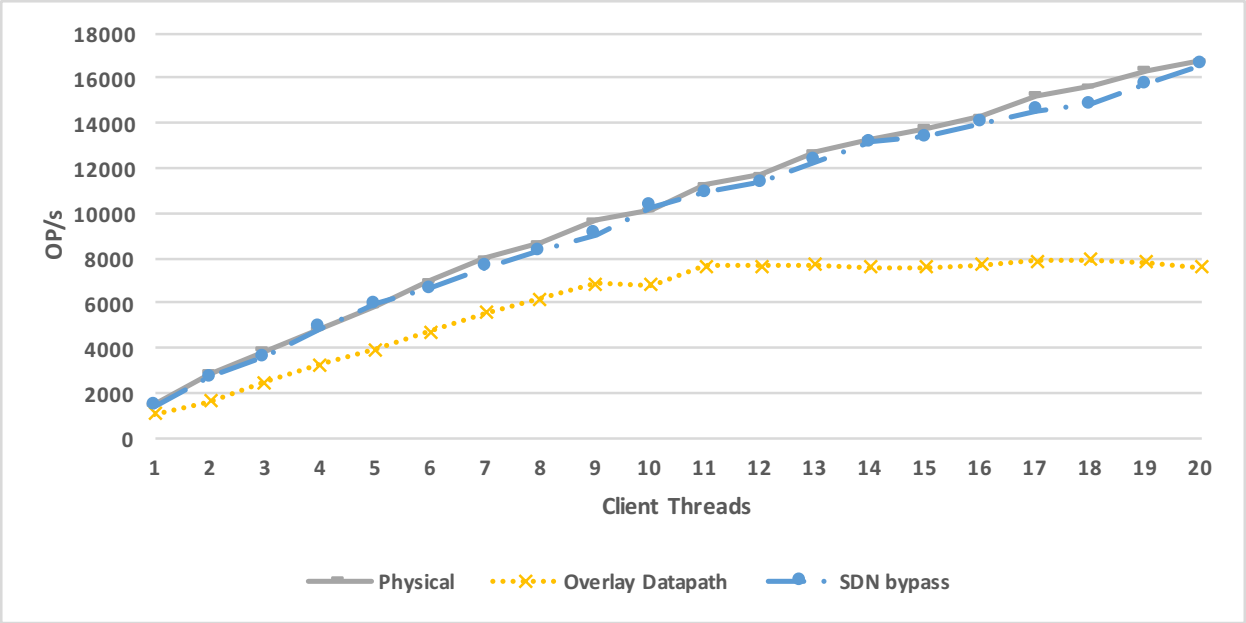

Figure 2-11. Redis simulation.

measured to be 13.5ms, 12.3ms and 2.63ms, respectively, in the CC, AA and GG cases. This latency is measured using the difference between the first packet and the last packet in the encapsulation path of each TCP stream.

2.6.3 Application Test

In this section, virtualization performance at the application layer is elaborated, using a round-trip latency sensitive application: Redis, a NoSQL, memory-based and key-value data structure storage system that is widely used. In the experiment, two host VMs are deployed in the CloudLab IG-DDC Cluster, provisioned with 862MB of memory. The test is done with the version 3.0.6 of Redis. The key length is 10 Bytes long and value length are set to 50 Bytes long, a common usage pattern of Redis [21]. For each run, clients make 1 million queries (50% sets and 50% gets). Every set packet size is 93 Bytes, and get return packets from the get are 36 and 57 Bytes, respectively.

Figure 2-11 shows the results of physical, overlay datapath, and VIAS SDN bypass cases. One important observation needs to be made – that the "Physical" case is one where when both Redis server and clients runs on the *host* VM, while in overlay datapath and SDN bypass

cases, those run on *nested* containers. Each thread setup a single TCP stream with Redis server. As the thread count increases, the throughput also increases, saturating at around 20 threads for LXC containers.

The throughput of VIAS SDN bypass is on par with the physical case, proportionally increasing the throughput as physical case with the increasing number of thread counts. On the contrary, overlay datapath throughput saturates around 8 KOP/S. After 20 threads, performance starts to degrade; this is due to not only overhead of SDN processing in the switch, but also due to resource limitations within the containers. The experiment shows that VIAS bypass performs significantly better than overlay encapsulation, and that it is capable to bypass multiple TCP streams simultaneously.

2.7 Related Work

The idea of building user-level virtual networks for Grid/cloud computing dates back to systems including Violin [46], VNET [86], and IPOP [47]. Violin proposed overlay virtual networks providing the abstraction of separate layer-2 networks for tenants. Violin was implemented without requiring any modifications on the VMM or the hosting network infrastructure, not only providing flexible user configurable network environment, but also reducing the threat of security risk from the host. VNET addressed a similar problem, focusing on the ability to inter-connect virtual machines across providers. This was accomplished by running VNET "proxies" on endpoints (e.g. VMs) at different sites, and tunneling L2 traffic over TCP/TLS links.

A limitation of these systems is performance – nowadays, cloud network infrastructure performance is 1Gbps or higher, making it challenging to deliver high-performance network virtualization at the application layer on commodity CPUs. VNET/P [95] addresses the performance gap by implementing a kernel module for layer 2 virtualization of guest VMs inside the VMM increasing the performance of virtual networks substantially by moving virtualization process from user space to a VMM module. It showed that it can achieve line speed performance virtualization in 1 Gbps network and 78% in 10 Gbps network. However,

it requires changes to the VMM, which hinders deployment of this technique. VIAS seeks to also bypass user-level processing, but does so while reusing existing, unmodified systems by leveraging software SDN switches. The experiments have shown that it is possible to deploy VIAS in existing cloud infrastructures (Amazon EC2, Google compute engine, CloudLab) without requiring any changes to VMMs nor VM images – VIAS only requires user-level software to be installed.

Another approach to minimize context switch between kernel and userspace is Netmap [76]. It eliminates the copy operation by using shared buffer and metadata between kernel and user space, showing that it achieves 20x speedups compared to conventional APIs. However, netmap relies on its own custom kernel module, again hindering the ability to deploy on commodity systems.

Instead of providing full virtual network to the users, VirtualWire [93] takes advantage of connect/disconnect primitives, which are assumed to be exposed by the cloud service provider. Their implementation requires changes to a Xen-blanket hypervisor, while VIAS can be deployed without changes to the VMM. Furthermore, VIAS makes no assumption about the primitives exposed by a provider to manage connectors – VIAS overlay links can be established even when cloud instances are constrained by NATs. They showed live migrating, with a given simple primitives, a VM instance from one network to the other successfully.

NVP [51] is closely related as it also shows combined version of tunneling and SDN fabric. However, this requires all the logical and physical network switches should be SDN harnessed. It provides REST APIs to the tenants to expose network virtualization capabilities. Network elements such as switches and ports are presented to the tenants, and tenants build topology of their network. Then, tunnels and flow rules are created and programmed by NVP to each hardware and software OpenFlow switches to forward packets among VMs deployed intra and inter clouds. However, NVP is designed to support multiple tenants in a single cloud provider. Unlike VIAS, its techniques do not support inter-cloud network virtualization.

2.8 Chapter Conclusion

Chapter 2 presented the novel architecture of VIAS, and demonstrated its ability to automatically provide fast virtual network paths within a cloud provider via coordinated programming of SDN switches, while retaining the ability to dynamically establish inter-cloud virtual private network tunnel. The main contribution of Chapter 2 is a novel user-level approach to distributed, coordinated control of overlay and SDN controllers, supporting private inter-cloud and high-performance intra-cloud network virtualization flows.

VIAS leverages existing system-level VMM/kernel software without modifications, and has been demonstrated to work by extending existing overlay software (IPOP) and SDN platforms (RYU, Open vSwitch) in realistic cloud computing environments, including Amazon EC2 and Google compute engine. Results showed that VIAS can provide flexible layer 2 virtual network, in particular to nested virtualization environments, where tenants deploy containers across multiple providers.

While this dissertation quantitatively evaluated the use of VIAS with software virtual switches, the use of the OpenFlow standard allows VIAS to tap into hardware SDN resources, if available.

CHAPTER 3
SDN FOR MULTI-TENANT DATA CENTER

As server virtualization technologies mature, multi-tenant cloud computing has become widely used as a platform for on-demand resource provisioning. Dynamic provisioning of computing resources as virtual machines provides flexible computing infrastructure that can be tailored by tenants, and provides high availability and proximity to clients by geographically distributing servers across different zones.

From a networking perspective, multi-tenancy in data centers requires an ability to deploy arbitrary virtual network topologies upon physical network infrastructure, along with managing overlapping addresses between different tenants. In addition, for better utilization and high availability, VM migration is an essential feature in Multi-Tenant Data Center (MTDC). When VMs are created upon user's request, they should be virtually connected to the user's already deployed VMs, regardless of their physical location. Also, when live migrated, the network identity (such as MAC or IP address) of VMs should remain the same. Finally, when VMs are terminated, their network presence should be revoked from the tenant virtual network, and network identities should be reclaimed. Tunneling has been widely used to tackle these issues by providing a virtual overlay network upon the physical data center network. Tunneling advantages include ease of deployment and separation from physical network topology, because it obviates the need for additional protocol from physical switches (such as VLAN or MPLS).

Generally, the tunneling process involves encapsulating every packet from the virtual overlay network by a physical network header. This process is usually achieved by general purpose processors at end-points and implemented in software (e.g. hypervisors), instead of physical network fabric. Nowadays, link speed of 10 Gbps is prevalent, and the trend is toward higher rates. While network devices surpass 10 Gbps, it is increasingly difficult for the tunneling process to keep this pace: the processing time of packet classification is dominated by memory access time rather than CPU cycle time [31]. Although current CPU technology achieves few

hundreds of GIPS(Giga Instructions per Second), the tunneling process hogs up computing resources at hypervisors, nibbling available resources for the guest machines.

Software Defined Networking (SDN) [57] has been widely used for implementing flexible routing rules against elastic traffic demand and also as a means to provide network virtualization in data centers. For example, NVP [51] uses forwarding paths with Open vSwitch, but its scope is bounded to a single hypervisor – it requires tunneling to reach beyond end-hosts. WL2 [14] leverages MAC address rewriting and a redesigned Layer 2 address scheme for hierarchical data forwarding, but it still depends on tunneling (VXLAN) and gateways for scalable hierarchical fully meshed Layer 2 network. VIAS in Chapter 2 uses transport address translation using SDN primitives to bypass selective traffic intensive TCP/UDP streams from encapsulation, while letting the overlay handling rest of network protocols.

In Chapter 3, PARES (PAcket Rewriting over SDN), a network virtualization framework for MTDC that leverages the OpenFlow SDN protocol, is introduced. PARES achieves network virtualization by packet rewriting on SDN fabric, without incurring processing overheads on end-host hypervisors. Experiments show that PARES achieve near-native virtualization performance at 10 Gbps link, while VXLAN-based tunneling achieves only 20% of line rate. Additionally, since PARES is implemented exclusively in data-plane fabric and packets are processed in an in-situ manner, it inherently has an advantage of performance isolation from the hypervisor, and avoids the overhead of layers of indirection caused by virtualization.

3.1 Background

3.1.1 Multi-Tenant Data Center (MTDC)

Tenants in a multi-tenant data center require access to the public Internet, while also being required to communicate among its VMs as if they were connected to the same LAN environment, regardless of location. Requirements for the MTDC network infrastructures include:

- Managing and isolating overlapping address spaces
- Support of VM migration for flexible provisioning

- Decoupling of virtual from physical network topology
- Ability to scale to large numbers of nodes

There are two existing general approaches to provide Virtual Private Networks (VPN) for MTDC networks. One approach is to micro-manage all FIB (Forward Information Base) of all network entities in the data center by establishing link paths from source host to destination hosts, e.g. by using IEEE 802.1q (VLAN) or MPLS [78]. These protocols use additional fields to route or to isolate sub-networks without intervention of conventional link layer or IP network protocol. However, VLAN approach has not been widely used to provide VPN in MTDCs, because it incurs complexity of configuration and scalability limitations: initial MTDC used IEEE 802.1q to virtual LAN for bridging and isolation [10], but suffered from scalability limitation. For example, in traditional data center [7], VLAN is used to provide broadcast isolation and to slice Layer 2 network. Layer 2 address space can be sliced by VLAN ID and each VLAN ID is assigned to the corresponding tenant for isolation. However, there is a limitation on VLAN ID of 12-bit width (4096 entries), which is way below the requirement of current typical multi-tenant data center. Typical data center ToR(Top of Rack) switches consist of 24-48 ports [55] and each server could possibly provision up to hundreds of VMs. Although, as a rule of thumb, a single hypervisor is not assigned VMs more than the number of threads on physical machines. Single rack with 40 physical servers can readily reach the limits. Moreover, configuring forwarding paths also bears scalability limitations because of the limited TCAM entries of switches. Typical low cost commodity switch known to have around 16k entries [27].

The other approach is to encompass virtual network packets inside physical network header, which is referred as tunneling, overlay network or encapsulation. Arbitrary network topologies and addressing schemes of the MTDC network can be realized with this approach. Furthermore, it can be accomplished using commodity switches without the requirement of additional protocol features. However, this approach comes with the cost of encapsulation overhead: the virtual network header is encapsulated by physical network header, reducing the

effective MTU (Maximum Transfer Unit), and packet-processing on general purpose processors incurs frequent context switching on the hypervisor. In Chapter 3, we address these two issues of this approach.

3.1.2 Related Works of MTDC Network Architecture

Several MTDC network architectures and data center network virtualization techniques have been proposed. To name a few, NVP [51] uses the Stateless Transport Tunneling protocol (STT) [28], VXLAN [55] and GRE as tunneling protocols at a service node (which resides in the hypervisor) to provide network virtualization. To avoid the encapsulation overhead at hypervisor, NVP introduced STT, which amortizes the encapsulation overhead by coalescing multiple datagrams. Although NVP is able to achieve line rate of 10 Gbps, it still hogs up CPU resources at both end-hosts and requires special feature in NIC called TCP Segmentation Offload (TSO). Since STT disguises encapsulation header as regular TCP header, there is no handshake or acknowledgment; thus, middleboxes such as firewall or load balancers in the data center need to be modified. Furthermore, a single packet loss can lead to drop of the entire STT frame, and significant performance degradation. When NVP uses GRE tunneling, it only achieves 25% of line rate at 10 Gbps and incurs more than 80% load at CPU.

VL2 [27] uses IP-in-IP [81] and the tunneling process is done at the ToR(Top of Rack) switches, but its evaluation is based on 1 Gbps line rate. NetLord [58] uses an agent on hypervisor for tunneling and defines their own encapsulation protocol.

PortLand [65] uses packet rewriting instead of tunneling, by translating pseudo MAC and actual MAC at ToR switches. However, it assumes IP Core/aggregation switches have longest prefix match upon MAC address. It uses MAC addresses in a manner similar to IP addresses in a multi-root fat tree data center topology [1]. Although it is supported in OpenFlow-enabled switches, it is difficult to expect commodity switches to have this feature, since longest prefix match is only implemented upon IP address but not flat addresses such as MAC.

SecondNet [30] uses "source based routing" by using MPLS label field. Every packet is encapsulated by MPLS field, which records every egress port of every switch on the path. Since

data center physical network topology is fairly static with small number of hops, the 20-bit width MPLS label has enough space to record all egress port numbers of every hopping switch. Since it is not based on encapsulation it achieves 10 Gbps line rate with fair throughput balance among VMs. However, it has scalability limitations in terms of the IP core switch hopping count, and requires MPLS feature on every switch at the core and aggregation layers.

Common aspects of aforementioned MTDC network architectures are that they assume low-cost commodity switches with simple routing protocol at IP core/aggregation, which use only a few FIBs at each switch [1, 7]. Multi-root fat tree topology is especially popular these days, since it can achieve fair bisection bandwidth without the requirement of high-end switches at core or aggregation layer by using redundant multiple routing paths. Moreover, it enables commodity switches to replace high-end switches, especially in the core layer.

Another common aspect is that MTDC architectures virtualize network at the boundary of the core fabric network and provides a mean to resolve "many-to-one" mapping of virtual address space to physical underlay address space either by multiplexing addresses or encapsulating packets. These functions reside in hypervisor, edge switches, or ToR switches. As examples, PortLand uses edge switches to map multiple actual MAC addresses to a pseudo MAC address. NetLord and NVP place agents at the hypervisor to perform encapsulation of virtual address by corresponding physical network address. VL2 uses IP-in-IP in ToR switches by encapsulating virtual IP address by physical IP address.

Above mentioned data center architectures in academia and industry along with their virtualization technique is summarized in Table 3-1. Overall, there is a clear trend that state of the art data center with network virtualization requires simple routing policy at core/aggregation layer, and expands its address space at the "edge" of data center network infrastructure. The virtualization primitives, such as address mapping and enforcing isolation, are handled at the edge. Rather than doing so at the end-point servers/hypervisors, our proposed architecture places SDN-enabled edge switches around the core network fabric (as in Figure 3-1) and exploits them for packet rewriting.

Table 3-1. Comparison of various network virtualization in multi-tenant or enterprise data center.

	Address mapping methodology	Virtualization body	line rate	Miscellaneous
NVP	Encapsulation (STT, GRE, VXLAN)	Service node at hypervisor	Achieved 10 Gbps using STT	
VL2	Encapsulation (IP-in-IP)	ToR switch	1 Gbps	
WL2	Hierarchical addressing, VXLAN(Inter data center communication)	Gateway	–	Inter data center architecture
SecondNet	Source based routing using MPLS	Port switching based source routing (PSSR)	10 Gbps line rate	Limited hopping count on switch fabric
PortLand	MAC header rewriting (Pseudo/Actual MAC translation)	Edge switch	150 Mbps at 1 Gbps	
NetLord	Encapsulation	Hypervisor	1 Gbps	

3.1.3 Layer 2 Semantics in MTDCs

To traverse packets from one server (one VM inside hypervisor) to another server (another VM inside another hypervisor)[1] , MTDCs typically use tunneling. Moreover, to support the broadcasting nature of Layer 2 network, an agent on the hypervisor can be used to handle broadcasting (such as ARP or DHCP) and multicasting protocols as multiple unicast messages with tunneling, or use IP multicast trees. For example, NVP [51] runs a service node at the hypervisor and uses multiple unicast messages to implement broadcast/multicast semantics. VL2 [27] invokes a directory service at the shim layer (which is layer on network stack of end-host O/S invoking RPCs to directory server) to resolve location of end-host address, and then tunnels the original and subsequent packets. VL2 suppresses layer-2 semantics

[1] In Chapter 3, server and VM are interchangeably used as the same meaning. Servers/VMs are virtual instances inside physical hypervisor.

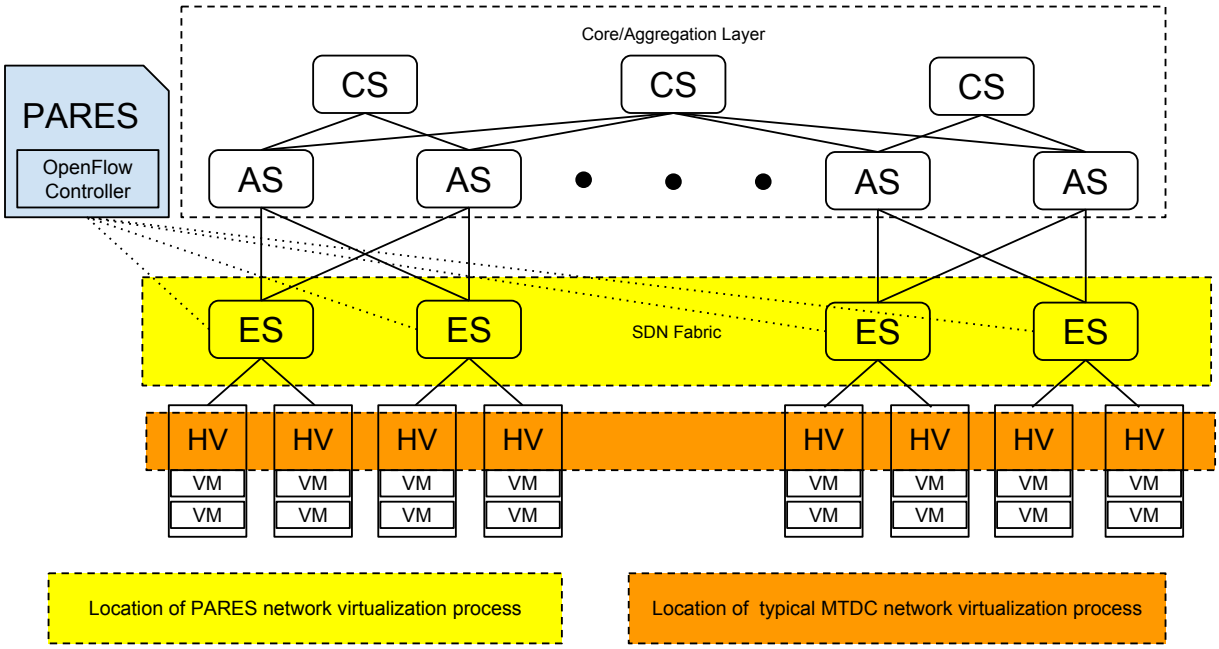

Figure 3-1. PARES in multi-tenant data center.

CS: Core Switch, AS: Aggregation Switch, ES: Edge Switch, HV: Hypervisor, VM: Virtual Machine

only at the edge switch layer to prevent scalability problems created by ARP. PortLand [65] uses proxy ARP, which forwards all ARP messages captured at the edge switch to the fabric manager. When initial broadcast ARP message is sent, it is intercepted at edge switch and is not egressed to the IP core/aggregation layer. Instead, the fabric manager (proxy ARP) itself sends this ARP message to all the edge switches in the network. PARES uses similar approach but it looks up its directory from the SDN control plane to resolve ARP. WL2 [14] intercepts ARP and DHCP packets from the virtual switches, then forwards to SDN control plane and the controller replies back to the end-hosts. It is similar approach with PARES, except PARES intercept broadcast packets at edge switches. As such, common principles of well-known MTDCs indicate that, although they keep the Layer 2 semantics at the edge (between network fabric and end-point hypervisor), they suppress it beyond the IP core.

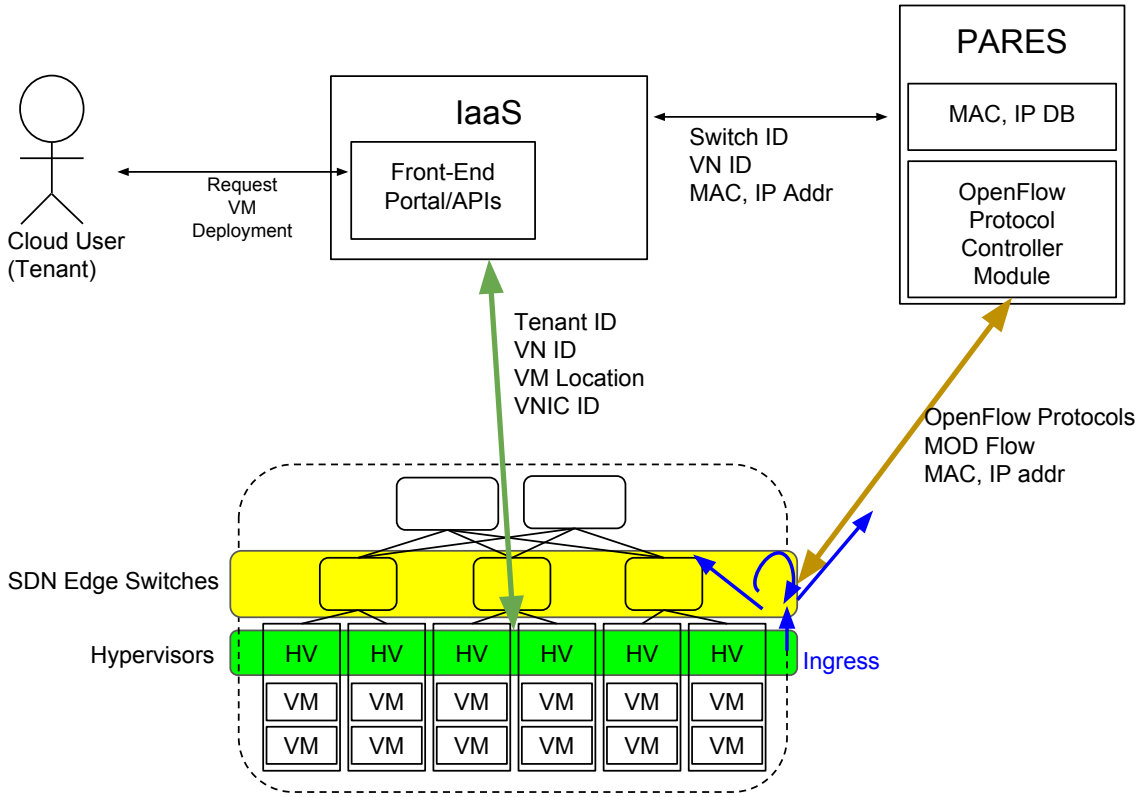

Figure 3-2. Architecture of PARES.

3.2 Architecture of PARES

3.2.1 High-level Overview

PARES resides between the front-end console portal/API and physical data center, as illustrated in Figure 3-2. Consider an example of a tenant requesting a VM instance deployment through the cloud service provider's front-end. The front-end processes authentication and access control, then virtual network configuration requests are sent to PARES through its APIs. PARES acts as an OpenFlow controller to program edge switches (Figure 3-1) dynamically, based on tenant requests. However, rather than statically programming each OpenFlow switches routing rule, PARES programs OpenFlow enabled edge switches dynamically, on demand, as shown in Figure 3-2.

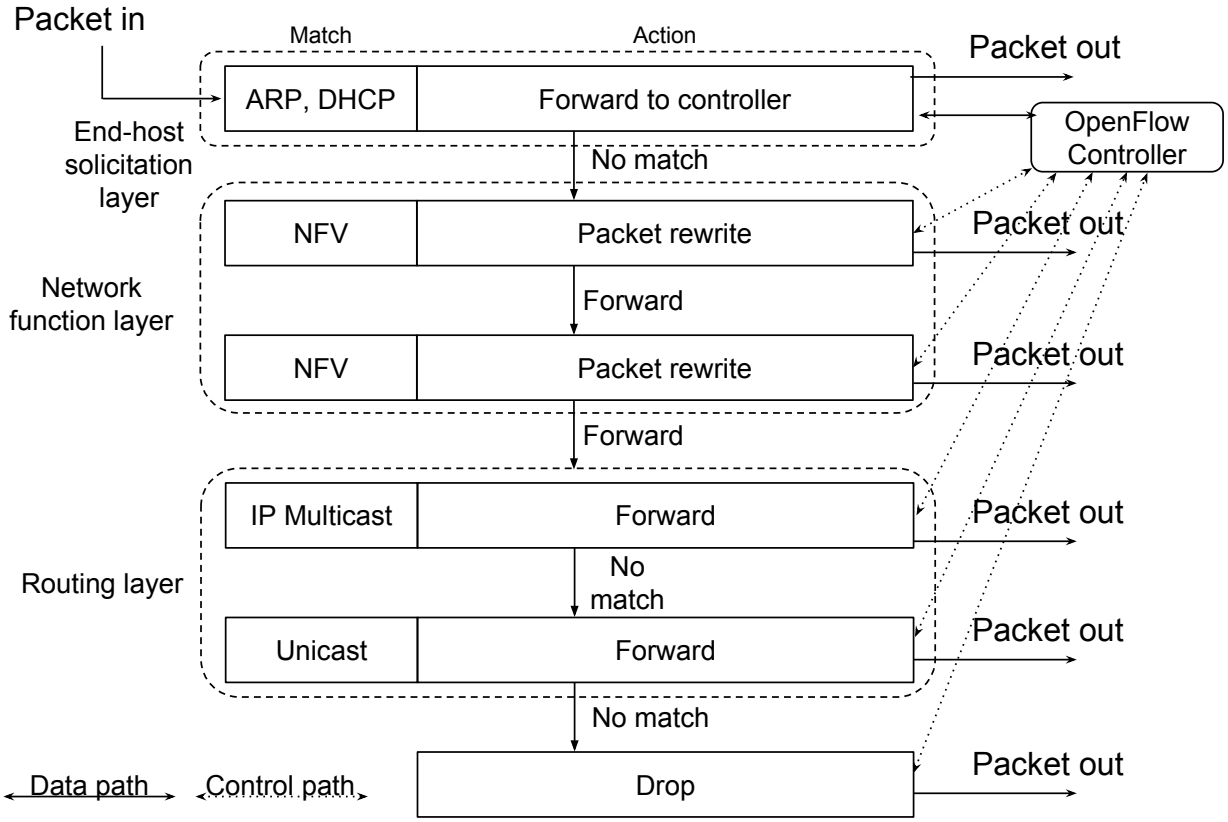

Figure 3-3. Operation layering on SDN fabric.

3.2.2 PARES Packet Handling Layers

Network layers of PARES should be different of the conventional OSI network layers. Conventional network stack has known to have hourglass shape, which have prevented adoption of new and advanced protocols because of its in-nature necessity of obeying the existing protocols. At each perspective of network layer, it does not have any knowledge or access of below or above network layer. Most of the PARES logic is for its OpenFlow protocol control module, which programs flow tables of edge switches (data plane) upon the given network configuration, and dynamically updates data planes upon the front-end's request. In essence, PARES programs the edge switches of SDN fabric in three layers: end host solicitation, network function and routing layer, as shown in Figure 3-3. The operation at these layers is elaborated below: In PARES, every layer has access to all kind of network header but layers are classified by its functionality, order and dependencies.

3.2.2.1 End-host solicitation layer

Separating names from location is a quintessential requirement for MTDCs, as providers and tenants require elasticity and VM migration. As explained in Section 3.1.3, MTDCs typically place a shim layer inside each server [27], or run service nodes on the hypervisor [51] to intercept location solicitation packets in order to prevent broadcast traffic from flooding the data center. VL2 [27] places a shim layer on every server in order to intercept location solicitation packets, then invokes its directory service (rather than broadcast protocol) to solicit location. This prevents broadcast traffic from flooding the data center. A somewhat different approach is used in NVP [51] where a service node at hypervisor creates individual tunnel for pairing remote servers. Broadcast traffic does not go beyond the host to physical data center network; rather it is replicated and tunneled by this service node. Different from these approaches, PARES places these functionalities of end-host solicitation on edge switches in the network fabric. This layer is the first layer a packet encounters when ingressed on the switch. If the packet is matched to any end-host solicitation protocol (ARP, DNS or DHCP), whole packets are forwarded to the PARES OpenFlow controller, and PARES replies (by looking up its directory) and injects the reply back to the original port.

3.2.2.2 Network function layer

All traffic other than end-host solicitation is forwarded to this layer. PARES places the network function layer before the routing layer, such that packet rewriting can be completed before routing. For example, in case of load balancing, destination transport address usually needs to be rewritten at network function layer before its final routing at routing layer. It is important to note that network functions can be independent or require sequential order among each other. For example, network traffic from north-south traffic first requires firewall then load balancing. Network functions are diverse in nature [12]; in this section, the scope of network functions is limited to only multi-tenancy (tunneling and address isolation). After processing each network function, the packets are passed to the routing layer.

3.2.2.3 Routing layer

After the packet modification in network function layer, switching/routing decision is made in this layer. If the underlay network is Layer 2 fabric, this layer performs mac-learning process, and switching is performed based on MAC address. While flood-and-learn semantics of Layer 2 nature provides easy deployment, STP (spanning tree protocol) prohibits exploiting possible redundant multiple parallel paths to increase overall bisection bandwidth. Although TRILL [18] and LISP [20] can be solutions for exploiting multipath in Layer 2 data center, it does not solve the problems of limited scalability caused by the number of TCAM entries in switches. As a result, enterprise/multi-tenant data center networks have evolved to shift from Layer 2 to Layer 3 fabric, due the to reasons outlined below. With the introduction of the multi-root fat tree network topology [1], which places all switch and servers in Layer 3 and exploits the existence and predictable latency of multiple parallel paths amongst all nodes, a pure Layer 3 data center network topology has become popular, especially in massively large data center [7]. Additionally, it maximizes the bisection traffic in data center network using commodity switches without the requirement of expensive high capacity switches at the core. If Layer 3 fabric is used as an underlay network, switching is performed based on IP address.

3.3 Implementation

In this section, it is described in detail that how each operation layer of PARES handles corresponding protocols or network functions.

3.3.1 Layer 2 Semantics in PARES

As explained in Section 3.1.3, MTDC architectures provide a means to intercept broadcast packets to prevent flooding the whole data center. PortLand's proxy ARP approach [65] intercepts ARP at edge switches then forwards to proxy ARP. VL2 [27] also has similar approach, but it intercepts frames at end-hosts rather than edge switches.

PARES uses an approach inspired by WL2 [14]: it intercepts broadcasting-based protocols on SDN-enabled edge switches. Instead of letting the broadcasting-based protocol to reach its destination, edge switches handle broadcast traffic by: 1) forwarding packets to PARES

through OpenFlow control plane, 2) waiting for reply from PARES controller following its directory mapping lookup, and 3) finally injecting reply message back to original port. [2]

Note that broadcasting is generally used for soliciting end-host address, not for actual packet traverse. Although using control plane incurs substantial latencies and can be a potential bottleneck, the performance impact of broadcast frames is typically negligible compared to actual data traffic (unicast) of typical MTDC applications. For example, ARP cache evicts typically in 60-seconds timeout and DHCP lease time is an order of minutes. Given the analysis from PortLand [65] and the evaluation in Section 3.4, a single core server can service ARP for several hundred thousand. Also, end-host address mapping information is rather static, with very few updates: it is only updated when a VM is created, terminated, or migrated.

3.3.2 End-host Solicitation Layer

ARP and DHCP are the most widely used protocols to solicit end-host network configuration. However, they flood the network due to their broadcast nature. These protocols are not involved in conveying the bulk of the (unicast) data traffic among end-hosts that their best interests would reside as close as to the end-host. Although, Layer 2 is supposed to link-to-link and Layer 4 for end-to-end transport, late appearance of DHCP make it place in the Layer 4 and above ARP and DHCP are Layer 2 and Layer 7 respectively, but DHCP packets can be inferred from Layer 4 header (transport port number).

End host solicitation works as the delegate of remote host to solicit ARP or DHCP server. End host solicitation in PARES directly handles ARP and DHCP using the approach described above (in Section 3.3.1).[3] It is important to place end-host solicitation protocols such as ARP,

[2] In OpenFlow, when packets are forwarded to the controller, it can read any fields of the packet. And the controller itself has a complete degree of freedom of crafting and injecting packets.

[3] Simple OpenFlow rules of forwarding all Ethernet type 0x0806 to controller (ARP) and transport port number 67 (DHCP) can achieve this.

NDP (Neighborhood Discovery Protocol) and DHCP to as near as possible to the end-host because of the flooding nature of those protocols, and to separate these traffics from the network fabric. Thus, flood-based packets do not spread beyond the end-host solicitation layer. Other discovery protocols, such as LLDP (Link Layer Discovery Protocol) [53] or NDP (Neighborhood Discovery Protocol) [62] for IPv6, and name resolution protocols (DNS) can be implemented in the same manner.

End-host solicitation protocols are resolved in a centralized manner in the SDN control plane by looking up a dictionary managed by PARES. In multi-root fat tree data center network, where IP address are assigned statically based on its pod location, traditional subnetting is not used inside the data center fabric. When ARP or DHCP request comes out from the end-hosts (e.g. hypervisor), edge switch forwards it to the PARES and ARP or DHCP message is injected from the virtual switches to the originated switch port. PARES exposes APIs to update entries of the MAC-IP mapping directory, such that the cloud front-end can use these APIs to update the information of which instance associates with which IP or MAC address.

3.3.3 Network Function Layer

Various network functions, such as firewalling or load-balancing, can be implemented in this layer. OpenFlow not only provides a means to inspect a packet header from Layer 2 to Layer 4 (match), but also allows it to perform actions such as forwarding to a switch port or rewriting network headers (action). This capability is enough to implement 5 tuples based network functions. Implementing these network functions are readily available in the literature, such as gateway in VIAS [45], and load-balancer and firewall in SoftFlow [42]. In this section, we concentrate on multi-tenant network functions and address isolation (tunneling equivalent), which are key to providing multi-tenancy for guest machines.

PARES tracks information regarding network functions. For example, if PARES deploys multi-tenancy network function, associated information such as locations of edge switches,

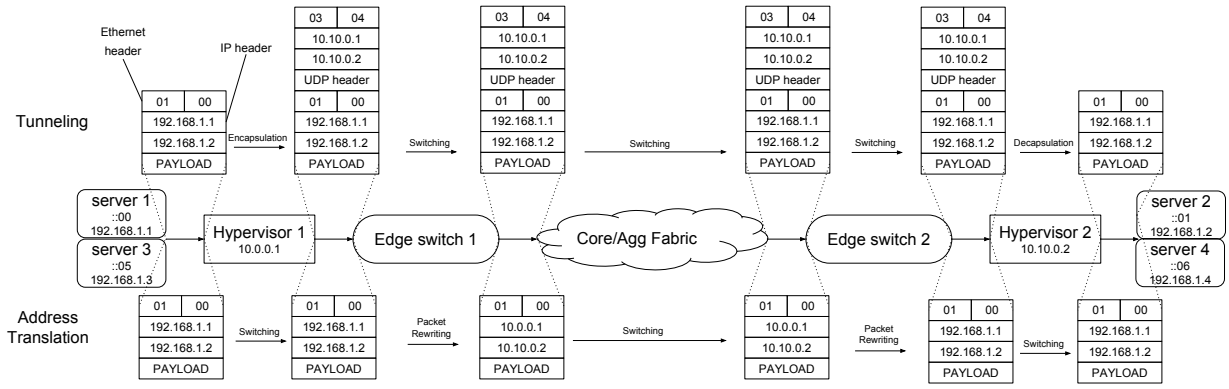

Figure 3-4. Datapath comparison of conventional tunneling (top) and address translation (PARES, bottom).

hypervisors or VM instances are stored in PARES to check the identifier conflicts (MAC address) or resource exhaustion (tables entries in edge switches).

3.3.4 Packet Walk-through Example

Consider an example where a tenant has two VMs (server 1 and server 2), and that server 1 sends a packet to server 2, which resides in the remote hypervisor across the data center, as illustrated in the bottom part of Figure 3-4. The tenant is provisioned the virtual network address range 192.168.1.0/24, and servers 1 and 2 are assigned virtual addresses 192.168.1.1 and 192.168.1.2. Edge switch 1 (which has been programmed by PARES upon tenant VM instatiation) uses an SDN primitive to translate the private IP address range to underlay network address range of the MTDC network fabric (10.0.0.0/8), then pushes the packet to the underlay network. The packet traverses the Layer 3 routing fabric, then reaches the destination edge switch 2. At this point, the edge switch inspects the MAC address of VM1, and translates the address back to 192.168.1.0/24. As a comparison, tunneling approaches encapsulate each IP packets (as shown in upper part of Figure 3-4), effectively reducing the MTU size by its increased header size and incurring frequent software interrupts on hypervisor. The role of PARES is in managing these rules on edge switches 1, 2, on-demand, based on tenant provisioning. Continuing the example, suppose there is a server 3 residing in hypervisor 1 (along with server 1), with MAC address 5. When server 3 sends packet to server 4 on

hypervisor 2, edge switch 2 translates back to tenant address range based on MAC address. Note that, while IP addresses are translated, the MAC address is not translated while the packets traverse the IP core/aggregation layer, since switching is solely done by IP address in the multi-root fat tree network architecture. This allows the translation at edge switch 2 to take effect.

3.3.5 Routing Layer

Conventional routing or switching based on IP or MAC address respectively is implemented in this layer. If the Aggregation/Core switches are based on conventional layer 2 network, this layer function as MAC learning and switching based on MAC address. If above switching layers are based on layer 3 as hyper scale data center such as massively scalable data center [15], this layer functions as longest prefix routing as multi-root fat tree architecture [1]. As a same manner, IP multicast tree can be constructed on this layer along with end-host solicitation layer. For example, IGMP subscribe packet is captured at end-host solicitation layer and PARES construct routing rule on routing layer.

3.4 Evaluation

In this section, we evaluate PARES from different perspectives. At first, since we apply above technique to a multi-root fat tree data center architecture, how to emulate this topology in the testbed is described. Then VXLAN is explained, which was used as the control group for experiments and performance comparison to PARES. Then, end-host solicitation layer of PARES is evaluated to demonstrate its ability to perform address resolution in MTDCs at scale. Additionally, it is shown that PARES can achieve line rate virtualization without using the end-host computing resources. Finally, scalability with respect to number of flow entries at each edge switch is evaluated.

PARES is deployed in two testbeds. One uses a hardware OpenFlow switch with 1 Gbps bandwidth, and the other uses 10 Gbps NIC with multiple Open vSwitch modules deployed on a single physical machine. All these resources are based on PRAGMA-ENT SDN testbed [40]. The first testbed uses a single hardware OpenFlow switch of model PICA8 3295 and two

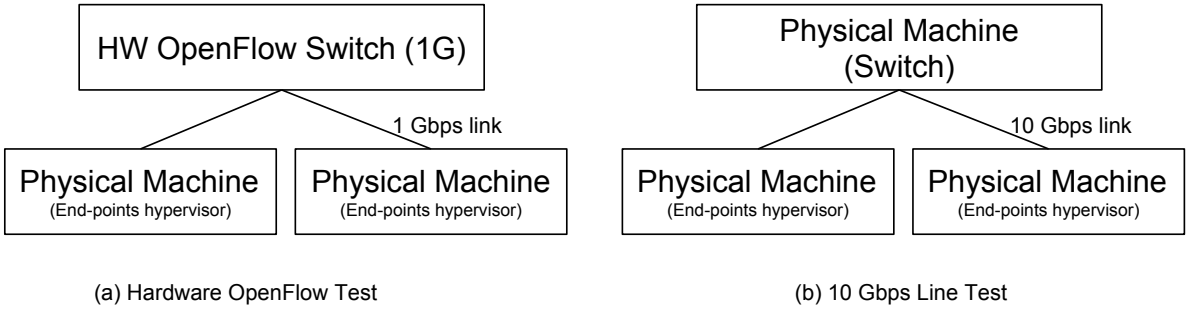

Figure 3-5. SDN testbeds.

physical machines (Intel(R) Xeon(R) CPU E5530 with 24GB memory) attached to it as shown in Figure 3-5 (a). The second testbed uses three physical machines, with the aforementioned CPU/memory specification, and with 10 Gbps Intel X520-DA2 NIC. To emulate the IP fabric core and edge switches, four Open vSwitch instances are instantiated in one physical machine, and use the two other machines as end-point hypervisors. All physical machines run Ubuntu Linux 16.04. Instead of using full or para virtualization hypervisor, lightweight OS level virtulization, Linux containers (LXC), is used, since the sole purpose is to evaluate network virtualization performance but not the processor virtualization. PARES has been built upon the open-source RYU SDN framework [6] conforming to OpenFlow specification version 1.3.

3.4.1 Multi-root Fat Tree and Bidirectional Longest Prefix

Routing decisions based on IP addresses are typically determined by longest prefix match. This can be used to make topology of Layer 3 network a tree-like structure, where the router closer to the top of the tree has shortest netmask, and those closer to the end-host have longer netmasks. This, however, introduces inherent bottlenecks and incurs severe over-subscription ratio at core and aggregation switches. The use of multi-root fat tree and bidirectional longest prefix routing schemes substantially reduce this problem [1]. Both most and least significant address fields are used on routing decision at aggregation/edge switches in the pod with two level routing table.

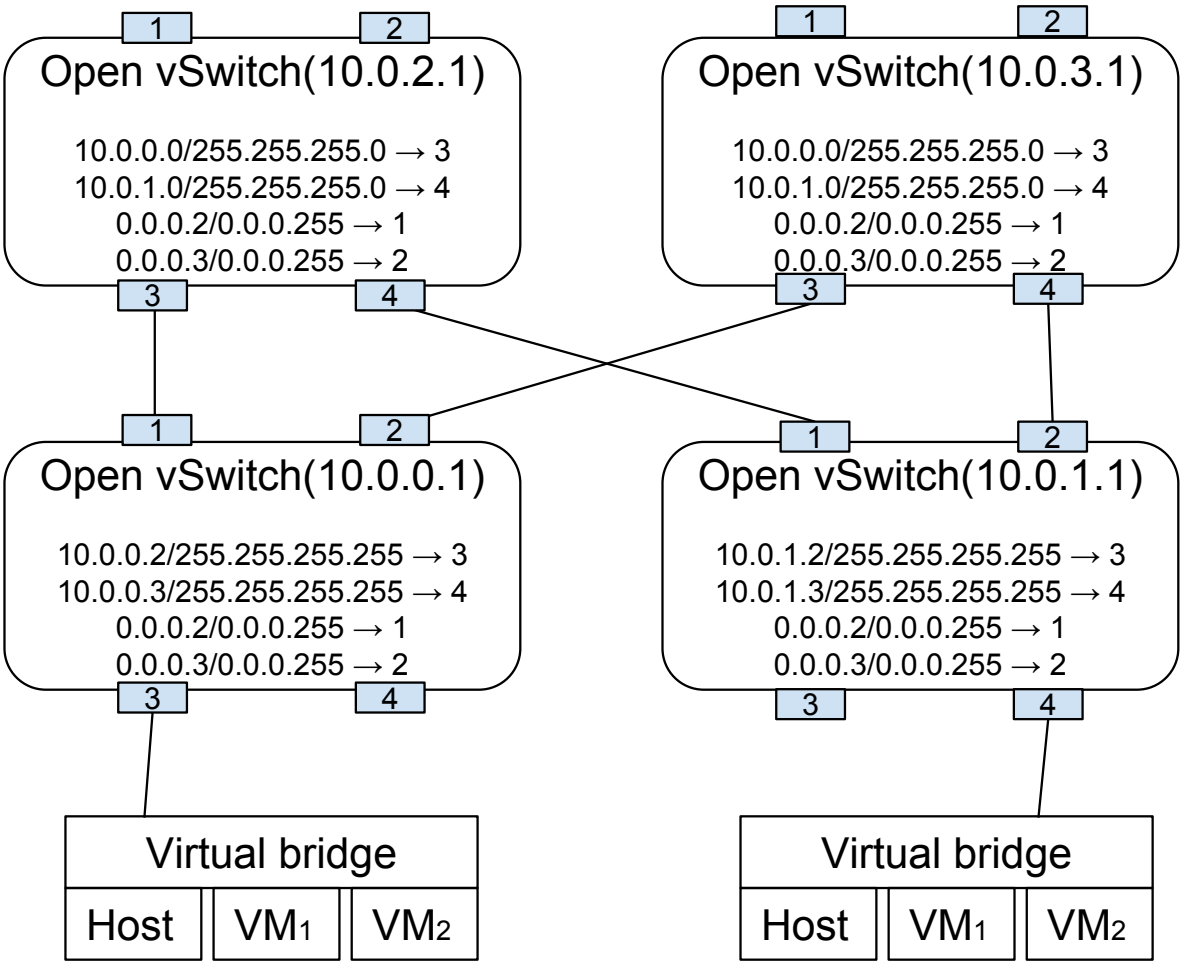

Figure 3-6. IP core/aggregation routing rule example with a single pod of multi-root fat tree data center architecture.

Bidirectional longest prefix scheme of flow entry is shown inside each Open vSwitch instances

For evaluation, as an underlay network topology and address scheme multi-root fat tree network topology [1] is used, which uses longest prefix match on IP address on either direction (from MSB to LSB or the other way around). OpenFlow currently supports this address scheme. While PARES is also applicable in conventional layer 2, or any other form of layer 3 data center network, the evaluation scopes only to system on a multi-root fat tree network topology. Specifically, a 4-port pod consisting of 4 switches is deployed as in Figure 3-6. Address range 10.0.i.1 is assigned for each switch (i is the index of each switch) and 10.0.i.j is assigned for each hypervisor (j is the index of each hypervisor). Note that 4 Open vSwitch

instances run on a single physical machine and the links between the switch instances are created with veth peer.

3.4.2 Virtual eXtensible Local Area Network (VXLAN)

There is a myriad of tunneling protocols available. To name a few, VXLAN [55], GRE, STT, IPsec Tunnel mode [48], NVGRE, and IPOP [47] have been used in various scenarios. In this section, we compare the proposed approach to VXLAN, which is a widely-used overlay network solution, especially designed for multi-tenancy in data center network. In addition, a kernel implementation of VXLAN is readily available and optimized for high performance.

VXLAN encapsulates packets from the tenants with the address range of underlay network. It has an extensive logical identifier space (16 million), enough for current multi-tenant data centers. To emulate Layer 2 flood-and-learn semantics in underlay Layer 3 network, VXLAN exploits IP multicast, which effectively reduces the range of flooding at the data center network. Because VXLAN does not require central control plane involvement nor additional feature from underlay network for its deployment, it is widely and flexibly adapted in current MTDC networks.

The limitation of the VXLAN protocol, as pointed out earlier, is that it takes up resources from the hypervisors. Every packet destined to remote tenants is encapsulated in an UDP header, which triggers software interrupts and multiple copy operation when crossing hypervisor-guest boundary.

3.4.3 End-host Solicitation Layer Scale Test

To test the end-host solicitation layer, ARP request messages are deliberately crafted from ARP traffic generator. As in Figure 3-7, we varied the rate of ARP requests per second from 1 to 1 million. PARES starts to exhibit latency increases at 10K queries per second. Given that typical ARP caches evict entries every 60 seconds, 10K per second is equivalent of ARP request from 600K end-hosts. A similar analysis is presented in PortLand [65] which runs proxy ARP server for all end-hosts. Note that this performance is achieved with a single SDN controller server.

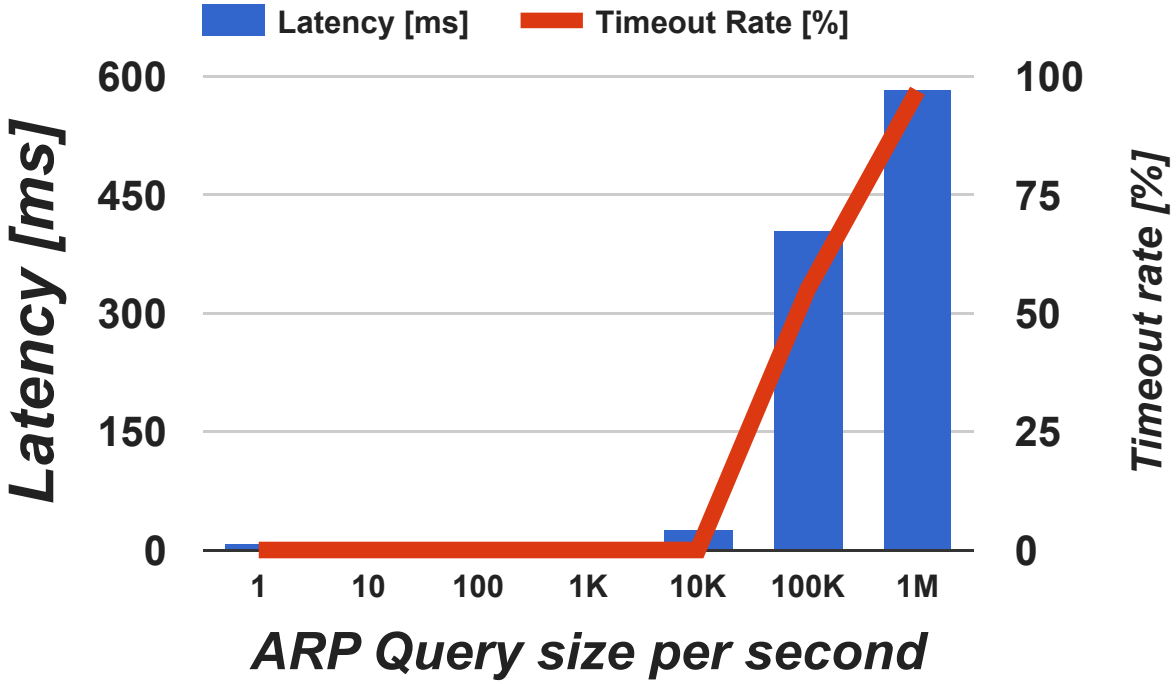

Figure 3-7. End-host solicitation test.

3.4.4 Line Rate Virtualization Test

Figure 3-8 shows the TCP throughput performance among underlay network native throughput (host), PARES and VXLAN using TCP performance tool "iperf". In the first testbed (1 Gbps), PARES achieves line rate virtualization while VXLAN peaks at slightly (about 4%) lower performance. This comes from the fact that VXLAN has reduced MSS size because of increased header size caused by encapsulation - the MTU is lowered to 1450. In the 1 Gbps testbed experiment, significant performance degradation caused by encapsulation is not observed, since there are sufficient computing resources for encapsulation.

Figure 3-9 shows the TCP throughput measured from an experiment using testbed 2. In this case, VXLAN only achieves about 20% of throughput, while PARES achieves near-native physical performance. Certainly, encapsulating at 10 Gbps places a burden on computing resources at end-point hypervisors and VXLAN could not keep the pace. This result is similar

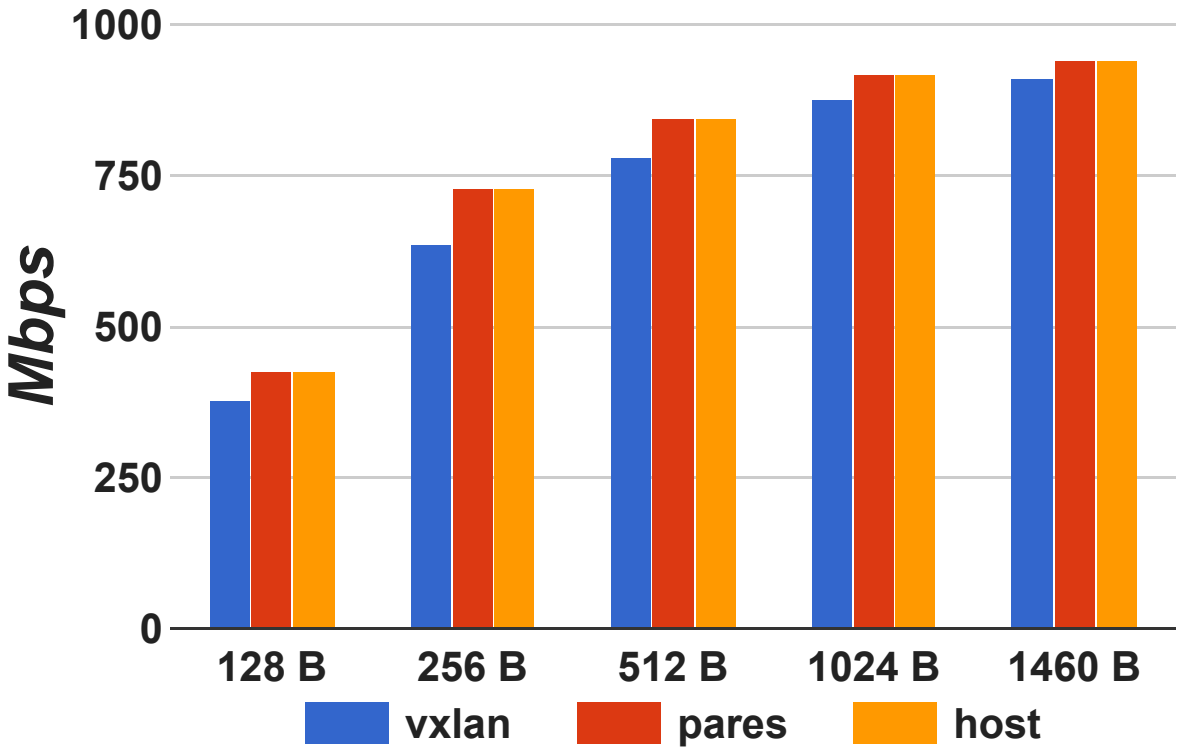

Figure 3-8. TCP maximum throughput comparison with various maximum segment size with 1 Gbps edge switch.

to NVP [51], which achieves only 2.4 Gbps with conventional tunneling (GRE) at 10Gbps line rate.

To confirm that the overloaded CPU is the source of the poor performance, CPU usage is tracked by using "top" command for 60 seconds of all physical machines (both iperf server(s)/client(c) and switch fabric machine(sw)) during the 10 Gbps TCP throughput test. Figure 3-10 shows the CPU usage profile for native physical case. It is observed that CPU usage is dominated by iperf at the server side. Software interrupt calls at both hypervisors are less than 1.0% of the time. Figure 3-11 shows the CPU profile of VXLAN at 10Gbps. It shows that the CPU profile is dominated by the interrupt calls ("ksoftirqd" at Linux kernel), and not by iperf. Note that VXLAN case achieves less than 25% TCP throughput of PARES or native case. Figure 3-12 shows the CPU profile of PARES. The trend is similar to native case, since it does not incur any interrupts at the hypervisor kernel.

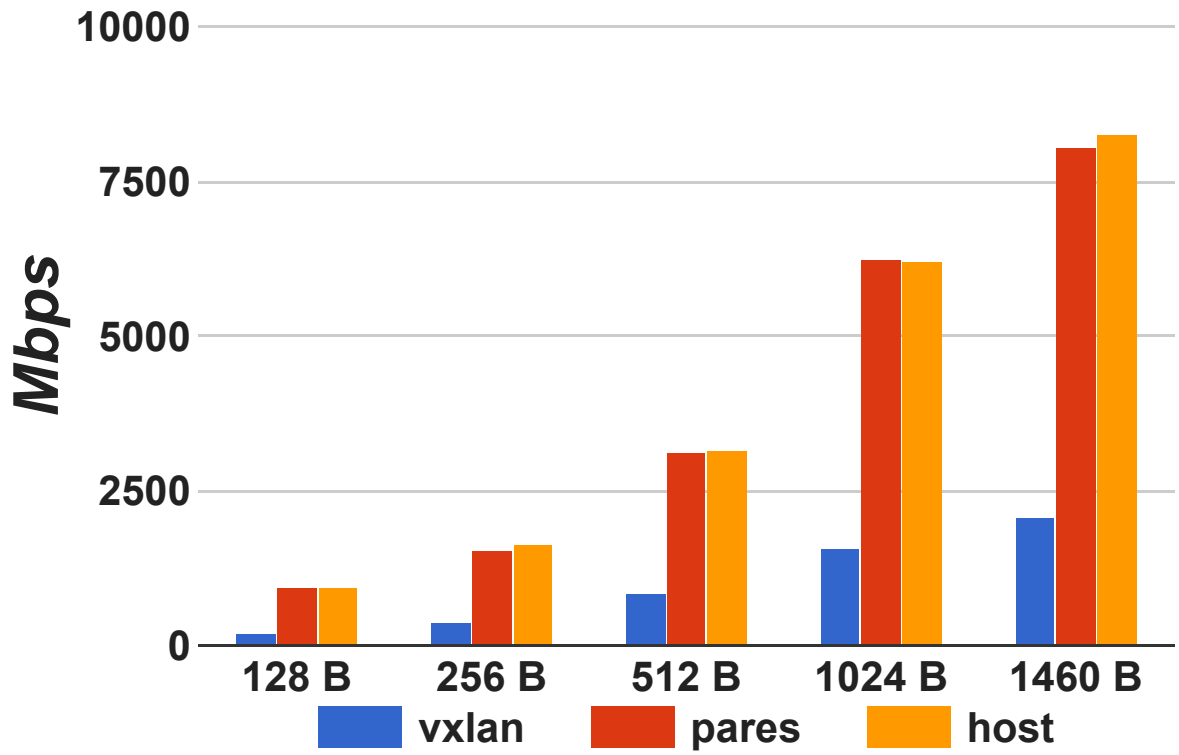

Figure 3-9. TCP maximum throughput comparison with various maximum segment size with 10 Gbps edge switch.

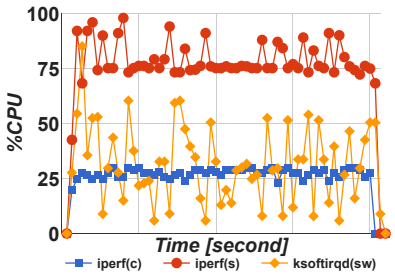

Figure 3-10. CPU profile span of 60 seconds run on physical hosts.

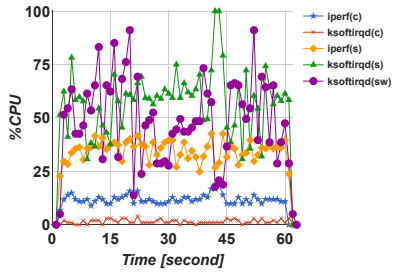

Figure 3-11. CPU profile span of 60 seconds VXLAN run.

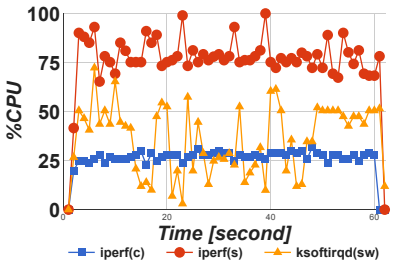

Figure 3-12. CPU profile span of 60 seconds PARES run.

In real data center, 99% of flows are smaller than 100 MB and 90% of traffic comes from flows between 100 MB and 1GB [27]. Single long flows can substantially benefit from PARES, and hence most of the traffic in data centers can benefit from PARES.

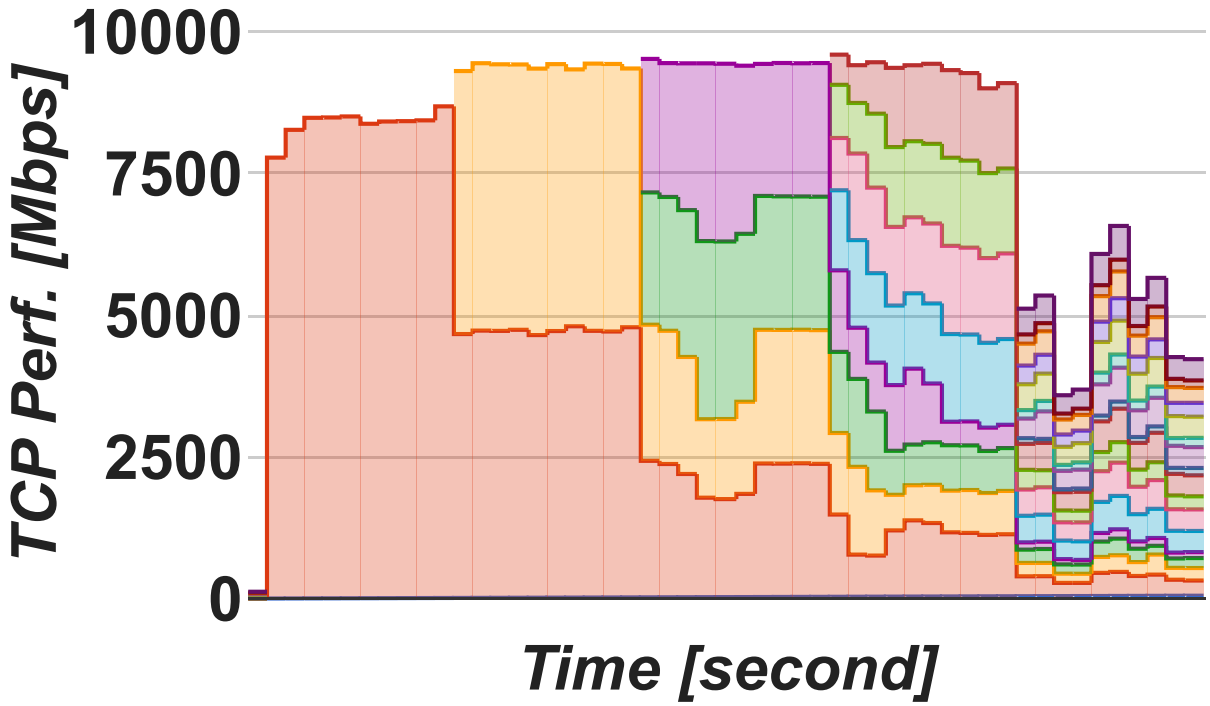

Figure 3-13. TCP Throughput among multiple VMs and streams.

3.4.5 Scalability Test

Figure 3-13 shows the TCP throughput when multiple VMs from each hypervisor start to run iperf at different times. The experiment is setup as follows: at time 0, only a single VM sends TCP traffic; every 10 seconds, additional VMs start TCP streams. Every 10 seconds, TCP streams from different VMs on the same hypervisor doubles. Note that each TCP stream shares fair throughput. At 40 seconds, the count of TCP streams from different VMs reaches 16. The physical machines have two sockets and each CPU has 4 cores; thus 16 VMs incurred contention on processing resources, reducing the overall peak throughput. Although over-commitment of multiple virtual CPU to a single physical core or thread is common, all VMs provisioned on a single hypervisor contending for network resource at peak throughput is extremely unlikely in real data center workloads. Through this experiment, it is confirmed that PARES provides peak throughput of 10Gbps with fair share among multiple streams.

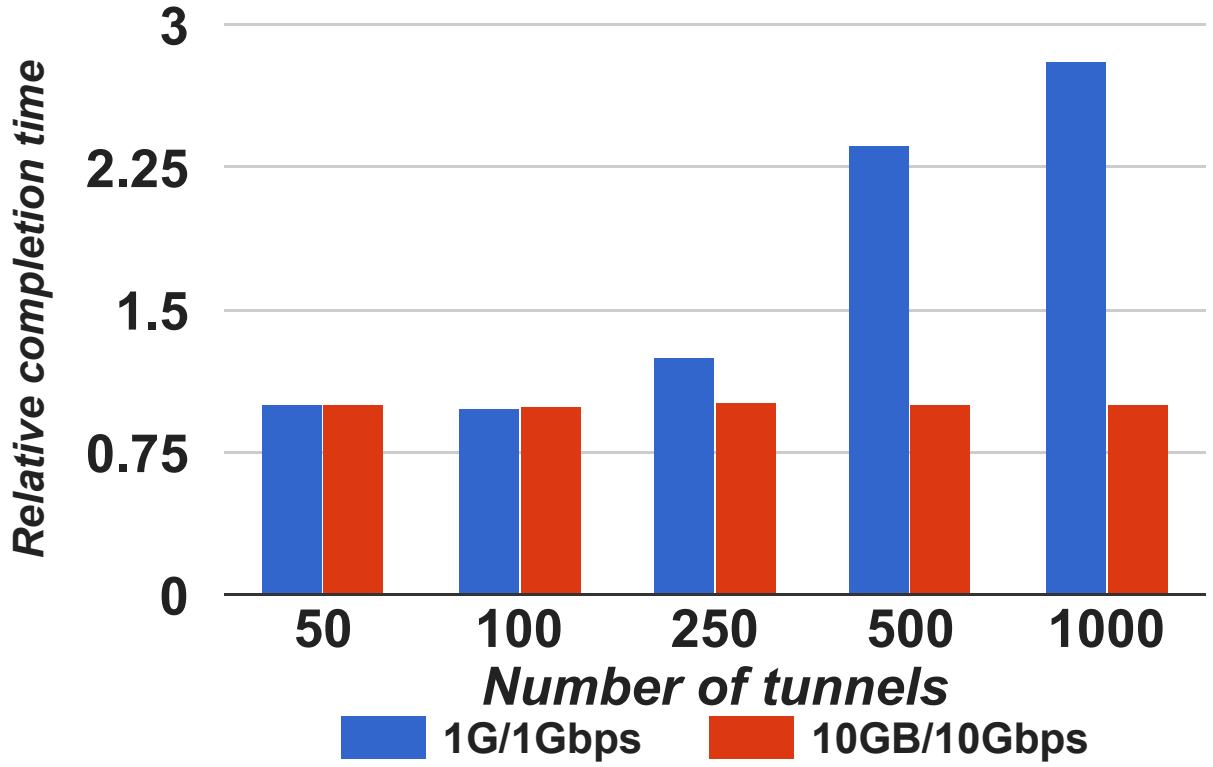

Figure 3-14. Flow entry scalability test (base values are 144s and 520s respectively for testbed 1 and 2)

Different from conventional MTDCs, PARES requires two rules per tunnel at each edge switch to perform multi-tenancy network virtualization. To test the effect of the number of flow entries, the number of entries at each edge switch are varied. Single long flow achieves peak throughput irrespective of the number of entries. Moreover, it is known that the majority number of flows are small (a few KB) [27] in real MTDC. It is therefore important to quantify the performance of PARES as the number of flow entries increases. Thus, TCP flow synthesizer is developed, of which based on real data center flow distribution from VL2 [27]. This traffic generator generates random size of flows with the distribution from real data center, randomly selects one of the tunnels, then transmits the flow using any thread available. The generator waits for an available thread if the number of iperf threads is more than the thread count. Note that when the number of tunnels is relatively small, it is more likely the generator chooses recently used tunnel, leading to less cache eviction in general

purpose processors. Two different transfer sizes (1 GB for testbed 1 and 10 GB for testbed 2) are generated and the completion time of the transfer is recorded in Figure 3-14. Since a single hypervisor cannot associate more than 1000 virtual network interfaces, the maximum number of tunnels in this tests is limited. In testbed 2, the number of flow entries does not affect the performance, while in testbed 1 it does. Initially, it is expected that the result would be the other way around, since testbed 2 uses a virtual switch and testbed 1 uses a hardware implementation of OpenFlow. However, while packet matching and switching is handled in hardware, packet rewriting is performed by a general purpose processor embedded in the PICA8 OpenFlow switch, which is a 850 MHz single core microprocessor. Although we observe a scalability limitation in testbed 1, this problem to be overcome with full implementation of hardware OpenFlow switch or faster multi-core microprocessors.

3.5 Discussion

3.5.1 Applicability to General Enterprise Data Center

Placing SDN devices at edge switches to separate IP core fabric and end-host network can also be applicable in general enterprise data center as a technique to scale out the number of end-hosts without changing the topology of underlay network or address/location consideration. Data center network architecture evolved in a way that allows to maximize bisection bandwidth by reducing over-subscription at core/aggregate layer with commodity switches [1, 27]. This trend sets forth as premise simpler routing rule at underlay network, while requiring network virtualization at the edge network devices [51]. Although we mainly discussed MTDC cases throughout Chapter 3, the same technique can be applicable in general enterprise data center. For example, VL2 [27] separates location address (LA) and application address (AA), which respectively represent Top of Rack(ToR) address (underlay network) and end-host addresses (application servers). It encapsulates AA by LA at ToR level, and those mappings between AAs and LAs are managed by the VL2 directory system. To handle broadcast such as ARP or DHCP, it places a shim layer at every end-host O/S network stack to intercept broadcast traffic and invokes directory service from data center. If we apply the

technique in VL2, shim layer is replaced with end-host solicitation layer from edge switches, and tunneling and its mapping are replaced by network functions layer. The approach of PARES does not require modification at end-host O/S network stack nor incur encapsulation overhead at network appliances at ToR.

3.5.2 Scalability Consideration

Given that Open vSwitch allows 1 million flow entries, it is safe to say that PARES implemented with Open vSwitch as edge switch has more than enough capacity to scale to typical deployments. Let us assume the scenario with 50-ports edge switch and a hypervisor at each port for a total of 50 hypervisors. Assume each hypervisor contains 10 VMs. In the worst case, all 5000 VMs belong to different tenants (although it rarely happens, since MTDC tries to co-locate VMs belonging to the same tenants on the same hypervisor.). If we assume 100 VMs per tenant, the total flow entries required is 100K.

The scalability in hardware OpenFlow edge switches can be limited by specification of corresponding hardware OpenFlow devices. Flow entries in typical hardware OpenFlow ranges around 100K entries. Although it is one tenth of the software one, it is enough size for above usage scenario. Moreover, hardware OpenFlow switch guarantees line rate irrespective of the flow table sizes, while large flow entries in software switches potentially incur cache evictions leading to performance degradation [70]. Currently, full implementation of HW OpenFlow switch with TCAM is costly. However, this problem can be overcome as SDN implementations become prevalent.

3.6 Conclusion

In order to provide multi-tenancy in data center or to extend Layer 2 semantics upon the Layer 3 network data center, "map and encapsulation" has been widely used in the literature and applications in industry. In Chapter 3, an alternative "map and address translation" approach is proposed, which translates network address from virtual range to underlay physical range using OpenFlow-enabled edge switches. PARES achieves line rate virtualization using this approach without incurring computing resource from end-hosts or hypervisors. Other

advantages of PARES are that there is no need for the modification on end-host O/S, nor requirement of computing resources from end-host hypervisor or servers, since it directly handles ARP and DHCP protocols on edge switches to separate Layer 2 semantic from the end servers to IP core layers. Experimental results show that PARES provides scalable line rate multi-tenancy virtualization at 10 Gbps without sacrificing end-host computing resources.

CHAPTER 4
OVERLAY NETWORKS FOR BULK DATA TRANSFER IN PUBLIC WAN ENVIRONMENT

Since the Internet is an aggregation of multiple ASes (Autonomous Systems), congestion control and utilization are not globally optimized. For example, it is not uncommon that a direct shortest route with low latency delivers less bandwidth than an alternative, long and roundabout route. Previous research has shown that geospatially distributed computing instances in commercial clouds offer users an opportunity to deploy relay points to detour potentially congested ASes, and as a mean to diversify paths to increase overall bandwidth and reliability. Such opportunity comes with a cost, as cloud-routed paths incur cost of not only provisioning of computing resources, but also for additional traffic to/from Internet. Well-established protocols, such as TCP, were created based on assumption of single end-point to end-point transfer; nonetheless, current computing devices have multiple end-points, and the increasing availability of overlay networks allows multiplexing multiple virtual network flows into a single physical network interface. In Chapter 4, we empirically evaluate the extent to which using cloud paths to transfer data in parallel with the default Internet path can improve the end-to-end bandwidth in bulk data transfers. In the evaluation, single-stream and multi-stream TCP transfers across one or more paths are studied. This dissertation also presents a design pattern as a pseudo socket APIs that can leverage the increased aggregate bandwidth of cloud multi-paths to increase overall throughput.

4.1 Cloud-routed Overlay Network (CRONets)

In CRONets [11], the authors ran extensive experiments involving 6,600 Internet paths, and observed empirically that using a cloud-routed overlay path can increase TCP throughput for 78% of the default Internet paths, with an average improvement factor of over 3 times. It uses overlay network such as GRE and IPsec to create virtual tunnel and the relay node to rewrite transport pairs using Linux iptables masquerade. While motivated by CRONets, we focus on a different study. Rather than experiments to assess alternate single cloud-routed path as a substitute for the default Internet path across large numbers of paths, various

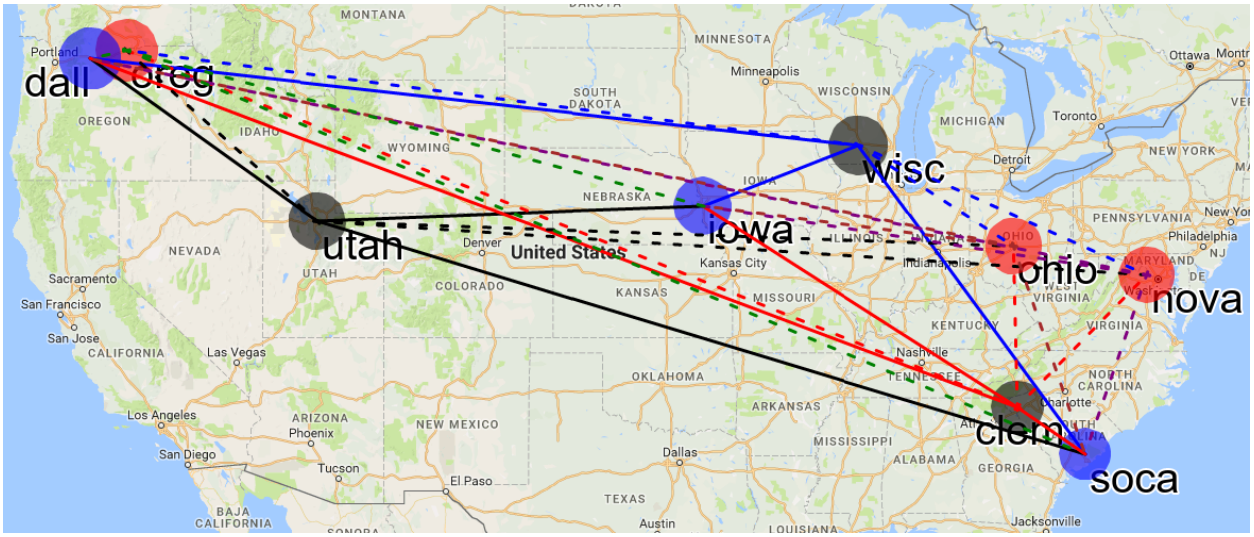

Figure 4-1. Locations of relay and end-point nodes

Relay nodes (red, Amazon EC2-hosted) and end-point nodes (black for CloudLab-hosted nodes, and blue for Google compute engine hosted nodes) are denoted as circles. Direct path is denoted as solid line, while detour paths relayed by Amazon EC2 nodes are denoted as dashed line.

experiments are conducted on how much aggregate bandwidth can be achieved by using multiple cloud-routed paths — in addition to, and concurrently with the default Internet path — and study socket level design patterns to exploit them to maximize the TCP throughput.

The experiments are performed to verify that the cloud paths considered in the evaluation (summarized in Figure 4-1) have equivalent bandwidth to the default Internet path, as in CRONets. The experimental setup is as follows. Three overlay relay nodes as Amazon EC2 small instances are deployed in availability zones in the west, mid-west and east (labelled "oreg", "ohio", and "nova"). Three nodes from CloudLab [16] (labelled "utah", "wisc" and "clem") and additional 3 nodes from Google compute engine (labelled "dall", "iowa", and "soca") are deployed as end-point nodes. These nodes serve as source and destination virtual machines in our experiments. The testbed is designed to explicitly address a requirement that traffic routes through the public Internet. While resources from CloudLab as endpoints are used in the experiments, traffic is not routed through Internet2 - rather, all traffic goes through the public Internet to the EC2 and Google compute engine nodes. CloudLab nodes

are not used as both source and destination endpoints in a measurement; this is to ensure that our experiments reflect the target use case of data transfers by commercial cloud users who may not have access to a research network with larger throughput and less contention than the public Internet. The setup leads to a total of 18 paths. As one example, consider the case where the source node is "utah" in CloudLab and the destination node is "soca" from Google compute engine. These two end-nodes have 4 Internet paths between them: the default Internet path, and three cloud-routed paths: relaying through "nova", "ohio", or "oreg" respectively.

Our measurements on this setup show that 6 out of 18 paths have at least one better alternative path than direct TCP path in terms of TCP throughput; the improvement factor is 6.54 on average. While the fraction of alternative paths candidates (i.e. those with higher throughput than direct path) is smaller in this setup than what was observed in CRONets (owing to the smaller number of relay nodes considered), the improvement factor is significant.

4.2 Overlay Path

In this section, an approach to build overlay paths through cloud nodes using tunneling and NAT (Network Address Translation) is explained. The cloud relay node is configured to create a pair of tunnels, one connected to the source, and one to the destination. The type of tunnel created depends on constraints imposed by end nodes. One type uses GRE (Generic Routing Encapsulation). GRE is well-known, easy to deploy, and a connectionless protocol without handshake that provides a persistent tunnel between peers. However, GRE not only requires public IP address to be used, but also it needs to be supported by the cloud provider. Public IP addresses may incur additional costs, and not all providers support GRE; for instance, Google compute engine does not. IPsec is another type of tunnel that can be used when GRE is unavailable. IPsec is a connection-oriented protocol and uses UDP encapsulation, such that one of the end nodes can reside behind a NAT router. Its encapsulation header is larger than that of GRE, thus leading to reduced MSS (Maximum segment size).

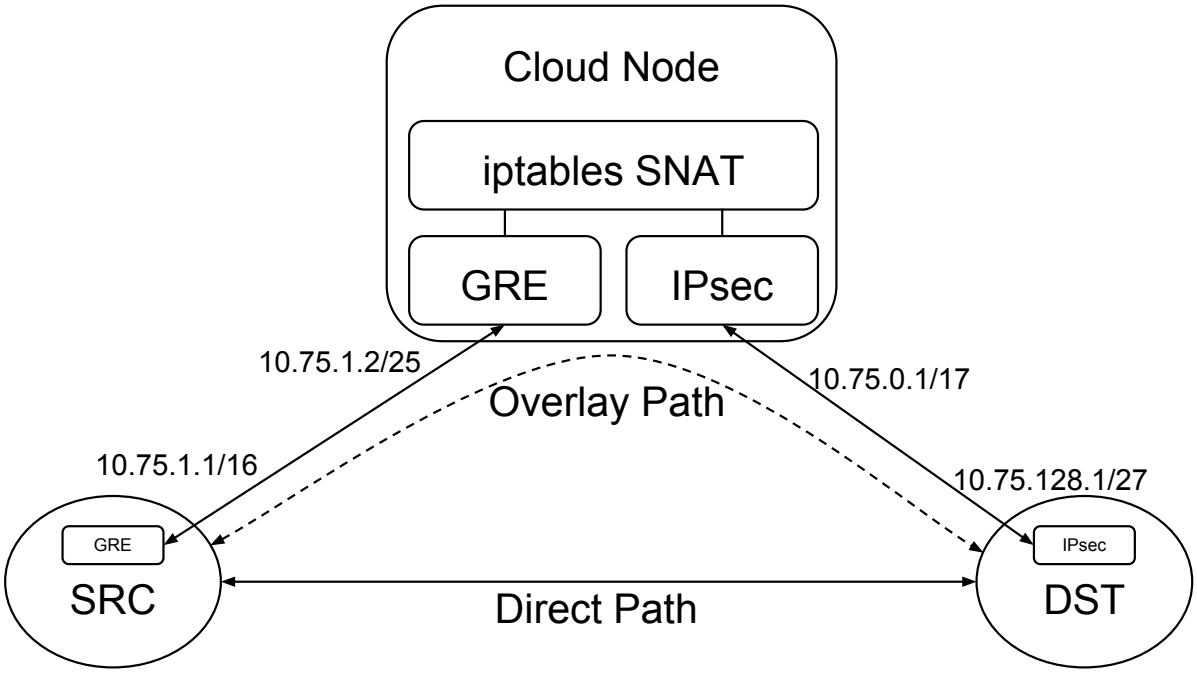

Figure 4-2. Overlay path relayed by cloud instances using GRE and IPsec tunneling.

Figure 4-2 shows an example of a single overlay path deployment with GRE and IPsec as the tunneling method for each of the end points. Note that GRE is a point-to-point protocol while IPsec is a site-to-site VPN. Thus, the subnet of local and remote end points of a GRE tunnel can overlap, while IPsec should separate subnets of local and remote address range. When the source node sends a packet to the destination, the cloud node rewrites source IP address to the next corresponding tunnel using Source NAT in Linux Netfilter. Linux provides a means to track connections and to conduct address translation in Netfilter framework along with the user-space tool iptables to set up the rules for it. When the destination replies to the cloud node, the cloud node rewrites the destination address to the original source node. In more detail: when the source sends to IP address 10.75.128.1, it first falls in subnet of GRE in the source, thus it is captured and encapsulated then sent to the cloud node. The cloud node, aware that the destination is over IPsec tunnel, rewrites the source IP address of the packet to 10.75.0.1. Meanwhile, Linux NAT tracks it by storing the 5 tuples of the stream (IP address and transport number pair for TCP/UDP). When the destination node replies to the source,

it first reaches the cloud node (10.75.0.1). Then, the cloud node, aware that the destination 10.75.0.1 is originally 10.75.1.1, rewrites the destination field accordingly.

Overlay path inherently incurs overheads compared to the direct path. Since both GRE and IPsec encapsulate original packets with a header, it reduces MSS by 24 and 40 bytes respectively. Also, if multiple tunneling is used on the path, Maximum transfer unit (MTU) of all the tunnels should match with the smallest MTU on the path to prevent IP fragmentation. Another source of overhead is packet rewriting at the cloud node. Packet processing and classification with general purpose processors also incurs overheads [31]; these are not a bottleneck in the use case scenario of bulk data transfers across the public Internet. In the testbed in Figure 4-1, the overlay path between a red and a blue circle is an IPsec tunnel and the path between red and black nodes is a GRE tunnel.

4.3 Finding A Stream Count Combination for Maximum Aggregate Bandwidth

In WAN environment, transferring data over multiple TCP streams can increase the aggregate throughput, even if the streams share a single routing path of a given source and destination pair, compared to a single TCP stream [2, 32, 73]. For instance, a representative application is GridFTP [2, 39], which takes advantage of this fact by transferring file data across multiple TCP streams to achieve improved throughput. GridFTP takes the number of parallel TCP streams as a configurable parameter from the user; while predicting the optimum count of concurrent parallel TCP streams is in itself a complex issue [96, 82], as a rule of thumb, the aggregate bandwidth increases logarithmically as the number of parallel TCP streams increases, then saturates (or decreases).

In order to verify whether this behavior is observed in our testbed, the aggregate TCP bandwidth on a single default Internet path with various stream counts is measured by using iperf. The aggregate bandwidth of each path is normalized with its peak bandwidth, then are averaged of all 18 paths. Figure 4-3 shows the ratio of increase in bandwidth by adding additional streams on a single path. The ratio is defined as $\frac{Aggregate\ \ Bandwidth_k}{Aggregate\ \ Bandwidth_k - 1}$ where k is the number of additional stream counts. It is observed that the bandwidth gain by increasing

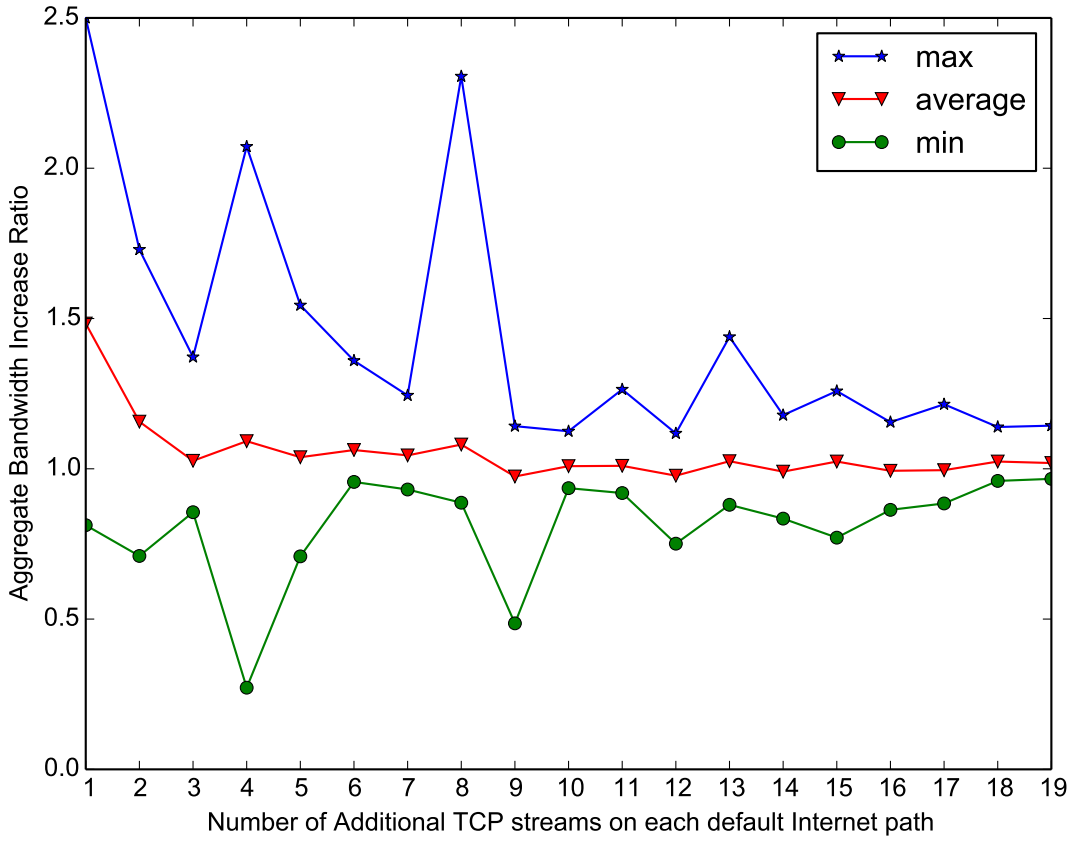

Figure 4-3. Average of relative aggregate bandwidth of the 18 WAN paths

stream counts is largest with small number of streams (four or less), and saturates around over 10. It is also observed that the aggregate bandwidth plunges in few cases as min line in Figure 4-3. Note that our testbed is public WAN that the aggregate bandwidth not only varies with path but also changes over time, as competing traffic do exist in public WAN or inside cloud service providers.

First 90 percentile stream count of the peak aggregate bandwidth is analyzed for each path. This analysis shows that 10 out of 18 paths fall in the 1-5 stream count bucket, and 5, 2, and 1 fall in the 6-10, 11-15, 16-20 buckets, respectively. Overall, a similar trend as in GridFTP [96] is observed, where aggregate bandwidth increases as the stream count increases until a certain point (usually below 10), after which it saturates.

Algorithm 1: Exhaustive search for finding a combination of number of streams for maximum aggregate bandwidth.

```
Data: path_set
Result: Maximum bandwidth with respective combination of stream counts on all
        paths
Function Wrapper(path_set)
    count_table[1, ..., path_set.length] ← {0};
    bw, Opt[1, ..., path_set.length] = FindMaxAggRec(path_set, 0, count_table, 0.0);
    return (bw, Opt);
Function FindMaxAggRec(path_set, path_index, count_table, prev_bw)
    if path_index > count_table.length then
        return (0.0, NIL);
    count_table[path_index]++;
    bw_np, Opt_np[1, ..., path_set.length] = FindMaxAggRec(path_set, path_index+1,
    count_table.copy(), max_bw);
    bw = Run iperf simultaneously on corresponding count_table and stores the
    summation of all bandwidth;
    if bw > prev_bw then
        bw_as, Opt_as[1, ..., path_set.length] = FindMaxAggRec(path_set, path_index,
        count_table.copy(), max_bw);
    return Max((bw_np, Opt_np), (bw, cost_table), (bw_as, Opt_as), key = lambda i :
    i[0]);
```

Knowing that the stream count that leads to maximum aggregate bandwidth on a single path varies over path to path, it is sensible that each stream count combination of each path also varies if the source and destination pair is multi-homed and has multi-paths between them. Finding stream count combination on multi-path has extra complexity, since paths are neither completely independent nor correlated with each other. The situation is completely circumstantial. It is possible that an alternate cloud path happens to share most ASes with the default Internet path, or that paths are completely independent from each other, depending on the endpoints. For the purpose of finding the stream count combination of multiple TCP streams on multi-path, Algorithm 1 is run on every source and destination pair in our testbed.

4.4 Algorithm Details

Algorithm 1 takes destination end-points with various paths as a path_set. The destination end points have a separate IP address space with a different routing paths.

The wrapper function initializes count_table array with zeroes and calls the recursive function "FindMaxAggRec". This recursive function calls itself with increased path_index, then increases count_table with given path_index and performs parallel iperf to measure the bandwidth with given combination of the count_table. If the measured bandwidth is larger than the previous bandwidth(prev_bw), it assumes that the corresponding path has more capacity to increase the bandwidth by increasing the TCP stream by one. It then recursively calls itself to measure the bandwidth with increased TCP stream by one. Note that every recursive call copies the count_table.

The complexity of Algorithm 1 is exponential: $O(N^M)$, while N is the optimal counts of TCP streams per path and M is the number of the available paths. In practice, exhaustively searching for the combination itself can be prohibitively expensive depending on the number of paths. Also, the combination changes over time as circumstances such as competing traffic in public WAN or inside cloud differs over time. It is also worth note that, in public WAN environment, finding global optimum of stream count combination is second to impossible. Even above exhaustive search Algorithm 1 potentially leads to local optimum as transient nature of public WAN make glitches at certain iteration that leads to forgo further recursion. The purpose of Algorithm 1 is to enable us to perform an empirical study to assess what are the possible gains in throughput by using multiple cloud paths. Heuristics to find a combination of paths/streams per path that strives to maximize performance are not within the scope of this dissertation.

The maximum aggregate bandwidth using multi-path found by using Algorithm 1 are shown in Figure 4-4, along with results of single stream on single path and multi streams on single path. By using multi streams on single path, aggregate bandwidth can be improved by a factor of 1.7 on average compared to single stream, while a 4.5 improvement factor on average can be achieved using multi streams on multi-path. In cases such as wisc-dall, soca-wisc and dall-wisc, where the default Internet path suffers from significant congestion, using multi-paths can substantially increase the aggregate bandwidth.

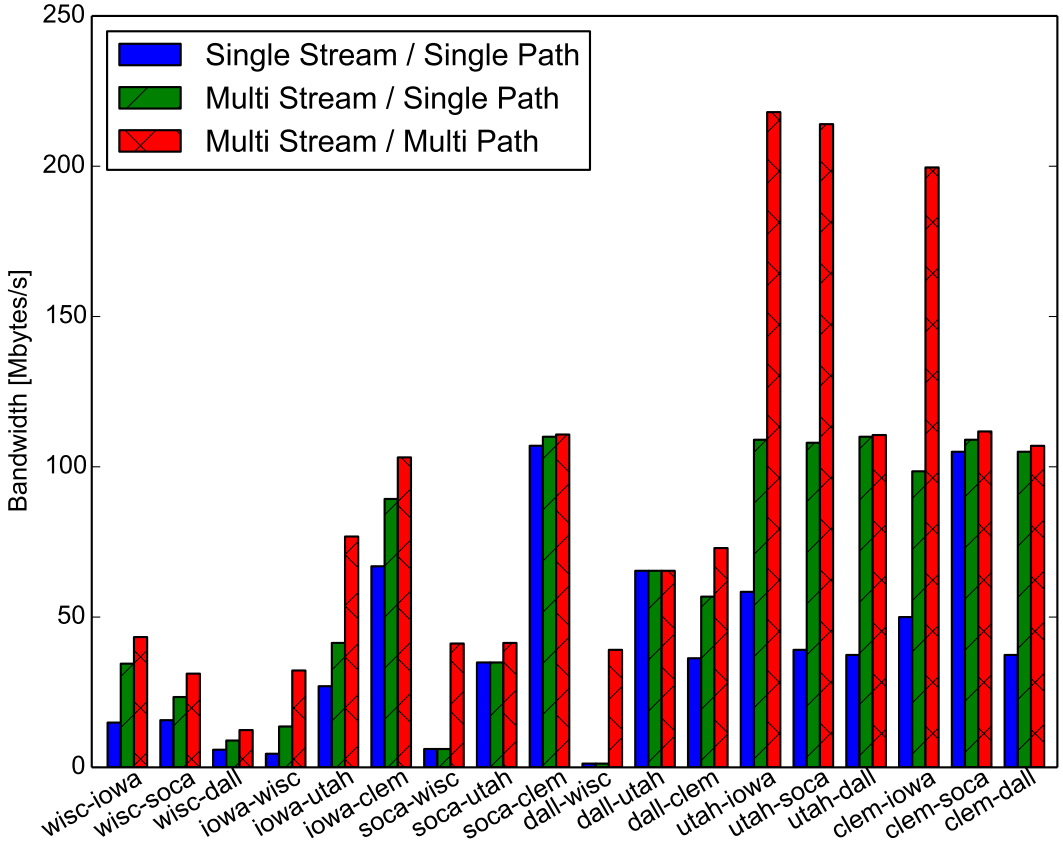

Figure 4-4. Aggregate bandwidth comparison among single stream on single path, multi stream on single path, and multi stream on multi-path.

To better represent the stream counts found in this experiment, Sankey diagram is used for their visualization (Figure 4-5). The streams flow from the left (sender) to the right (received); the line widths represent relative counts of streams. Lines hopping through relay nodes (oreg, nova, and ohio) represent streams that are routed through a cloud path, while lines directly connecting a sender to a receiver represent the default Internet paths. One notable phenomenon is that the oreg cloud relay, which is located fairly remotely from other nodes, happens to embrace more stream counts than the other cloud nodes. This trend also occurs in a Sankey diagram with aggregate bandwidth per path: oreg provides more bandwidth than the other paths. While the geospatial diversity of the cloud nodes in this test is limited,

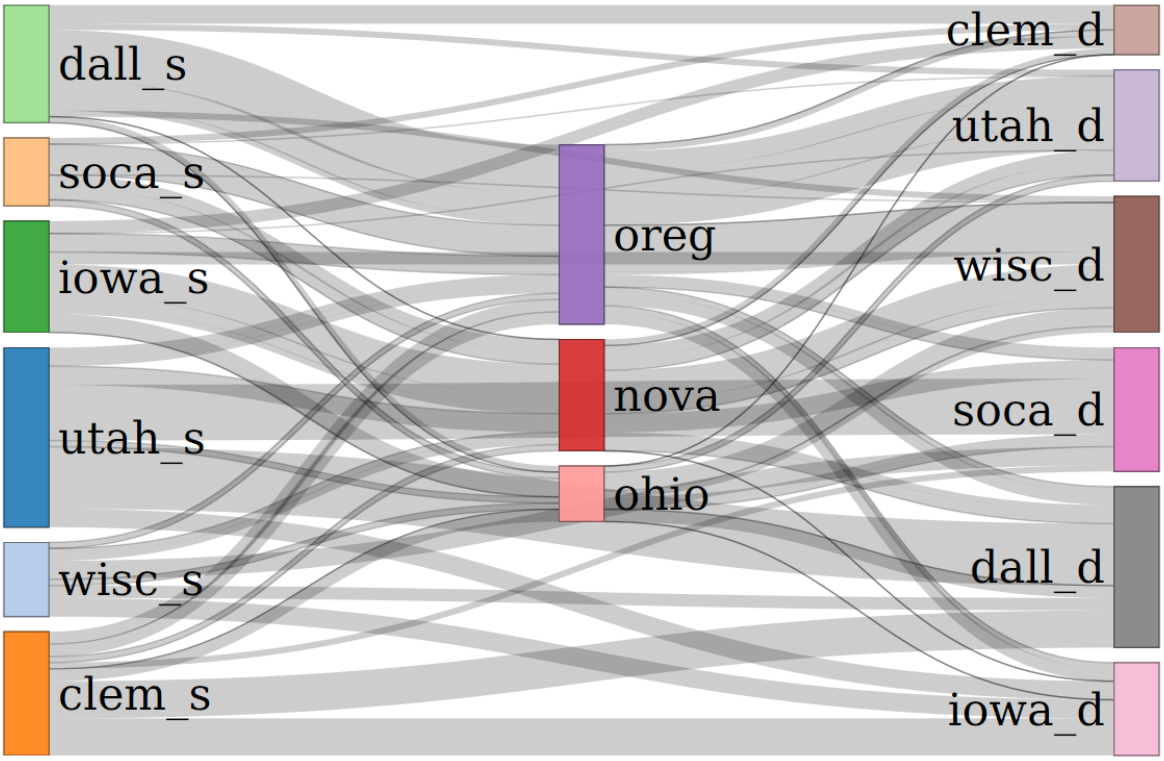

Figure 4-5. Sankey diagram from source(left) to destination(right) in terms of stream count. Line width is scaled with the stream count.

one possible explanation is that geographically distant nodes can provide more independent cloud paths, compared to the cloud paths located near the default Internet path.

Since our testbed is on public WAN environment, it is possible the experiment is affected by temporal competing traffic pattern such as diurnal or weekly pattern as shown in other researches [87, 23]. the experiment ran extensively to cover a week-long period, and frequently to cover the diurnal pattern. Rather than a regular pattern, sudden dips or spikes of throughput are observed occasionally in cloud path that are not correlated with other cloud paths, or the default Internet path. This as a factor from the zone of cloud service provider.

4.5 Parallel TCP Experiment Using Cloud-paths

Previous research [11] showed that a cloud routed path potentially has higher bandwidth than default Internet path. In this research, we target to increase the overall bandwidth with additional cloud-routed paths by simultaneously transferring packets using all available cloud

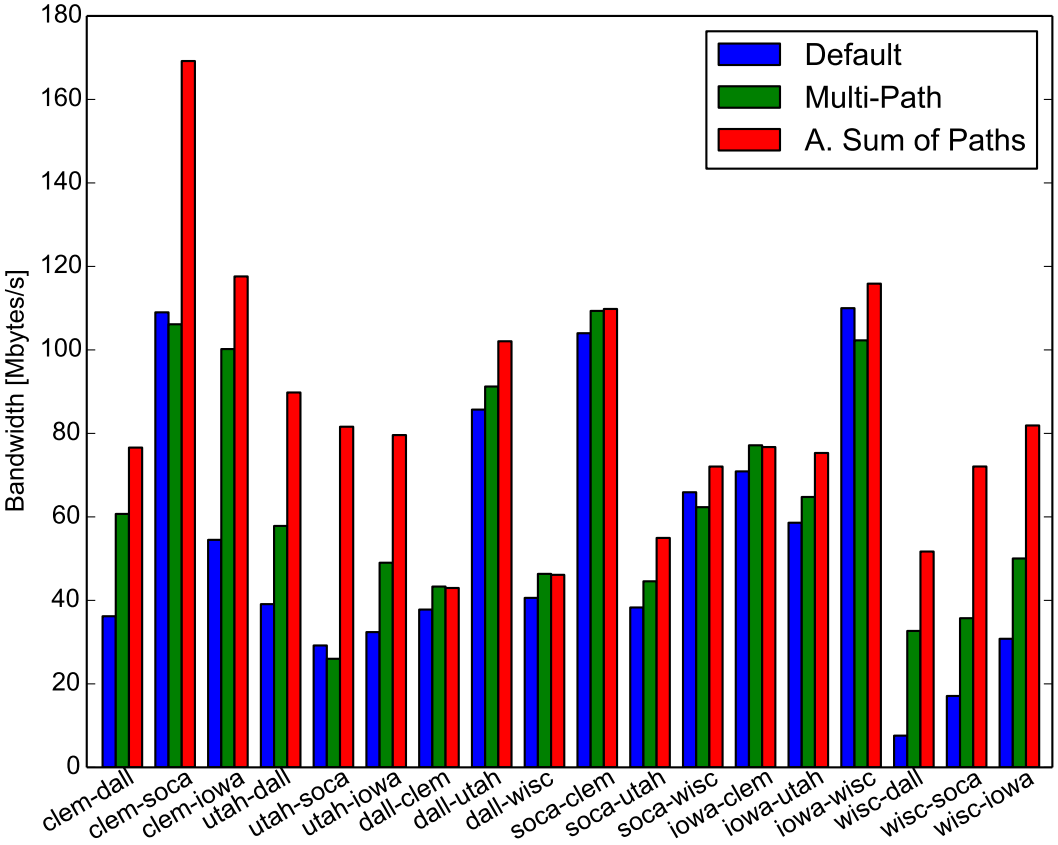

Figure 4-6. Bandwidth comparison among default Internet path, simultaneous multi-path and arithmetic sum of multi-path

.

routed paths along with default Internet path. To measure the bandwidth using multi-path transfer, TCP application called iperf ran all available paths simultaneously and compared the results with single default Internet path. In addition, iperf ran at all paths (defulat Internet path and cloud path) separately and added the bandwidth of all path. This arithmetic sum represents the theoretical maximum bandwidth, as cloud multi-paths happen to share the part of paths and to coincide in the same ASes. This test ran in all paths in our testbed and the result is shown in Figure 4-6. It is observed that using multi-paths can increase the chance to achieve higher bandwidth and the improving factor is 1.45 in average.

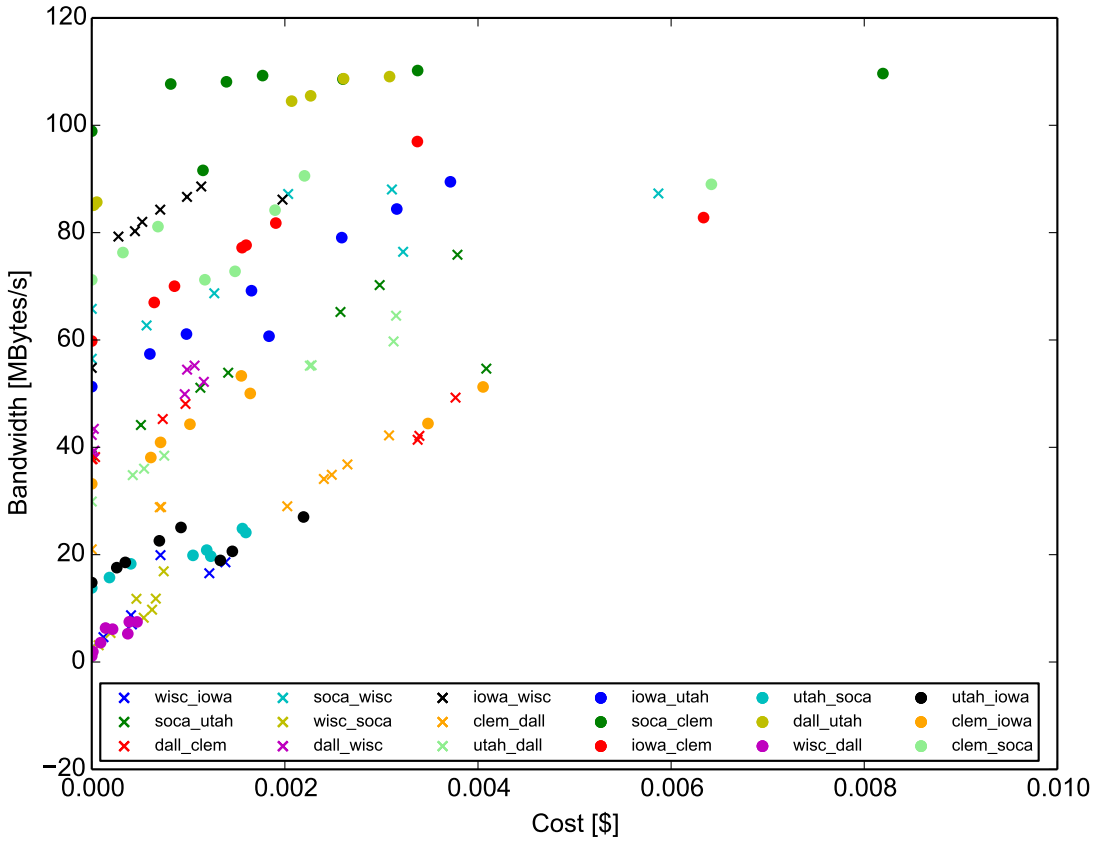

Figure 4-7. Bandwidth-Cost relationship (cost is for a 1GB data transfer)

4.6 Bandwidth-Cost Relationship in Cloud Multi-Path

The data analyzed in the previous section shows that cloud-routed multi-path can deliver higher aggregate bandwidth than default Internet path in several instances. However, unlike default Internet path, the cloud path incurs costs: both the cost of running a compute instance, and the network transfer cost. Provisioning cost is calculated based on number of hours an instance is running, and the traffic cost is calculated based on the number of bytes transferred out of the cloud provider. To characterize the relationship between bandwidth increase versus cost, TCP-based iperf is used to perform transfers with all possible combination of paths. In this experiment, a single path is characterized by running iperf on the path; to characterize multiple combinations of paths, iperf ran concurrently on all corresponding paths,

and add up all the data transferred. Since Internet paths have overlapping ASes among them, the multi-path bandwidth does not exceed the arithmetic sum of the individual single path bandwidth. The results from the characterization of bandwidth and cost in our testbed are summarized in Figure 4-7. As shown in Figure 4-7, dots on the left (y-axis) denote the default Internet path — it does not incur any cost. As more alternate paths are added to the default path, the cost (and bandwidth) increases. The data points on the right-most part of the graph use all available paths. The bandwidth/cost relation forms a convex curve; as the cost increases, the aggregate bandwidth increases, but in a diminishing manner. It is sensible that, as more alternate paths are used, there is a greater chance of overlapping ASes among paths. It is also important to note that potential increases in aggregate bandwidth are provided only in an opportunistic and granular manner, and incurring additional cost by adding up alternate paths does not always lead to higher aggregate bandwidth.

4.7 A Design Pattern for Multi-Path TCP Socket

In Section 4.3, we show that the aggregate bandwidth can be increased by using multiple streams on multiple paths. However, higher aggregate bandwidth does not always lead to higher end-to-end data transfer throughput. Packets are delivered out of order at receiving end, even in a single path, and in multi-path environment, out-of-order delivery at receiving end exacerbates as packets go through a different path with different RTTs. Widely used transport protocols, such as TCP and SCTP, handle this by retransmission or by sending gap acks back to the sender, but it leads to duplicate transmission and reduced congestion window at the sender [41].

In this dissertation, we do not introduce a transport level solution to handle out-or-order delivery. Instead, we slice the bulk data into blocks of sizes proportional to the respective bandwidth of the paths that the blocks are to be transferred over. Then, each block is sent over through a single TCP stream on an assigned single path, and multiple blocks are concurrently sent over multiple streams on multiple paths. Our-of-order delivery at the block

level can occur in this approach; it is the responsibility of the receiver side to reorder these blocks.

This approach runs Algorithm 1 to find the aggregate bandwidth of each path and the count of streams per path. Then the sender allots the total size of bulk data into each path by computing the ratio of the aggregate bandwidth of each path over the total aggregate bandwidth, such that the allotted size is proportional to the respective aggregate bandwidth of each path. Then the allotted size is divided evenly by the count of streams of the path, and the divided value becomes the block size.

To inform to the receiver about how many blocks are to be sent and on which path they are sent over, the sender and the receiver establish a socket connection for signaling. Thus, there is one signal socket to arbitrate data transfer between a sender and a receiver, and multiple data transfer sockets, matching the number of blocks. The signal socket between the sender and the receiver arbitrate signaling of beginning and ending of the data transfer, and metadata about the transfer session. In the prototype, "init", "ready", "done" and "fin" signals are introduced. Init signal from the sender notifies the receiver about the block counts for each path and that the start of data transfer is imminent. Ready signal from the receiver notifies sender that receiver has opened sockets and is waiting to be connected. It also notifies which port number is to be used for data transfer. Done signal from the receiver notifies the sender that the data transfer has completed and ready to terminate the session. Fin signal from the sender notifies the receiver that it has no more sessions, and closes the socket. In the implementation of the signaling protocol, typical TLV (Type-Length-Value) based approach is used, but without the type field. The length of all signal messages is shorter than two-byte integer range; the first two bytes are used as a length field, and the rest are used as a message payload.

Figure 4-8 shows the key elements of our proposed design pattern. The socket slices the data to be transferred into multiple blocks and queues to the data transfer thread along with block ID, offset and size of the block. Initially, before actually sending data, the socket

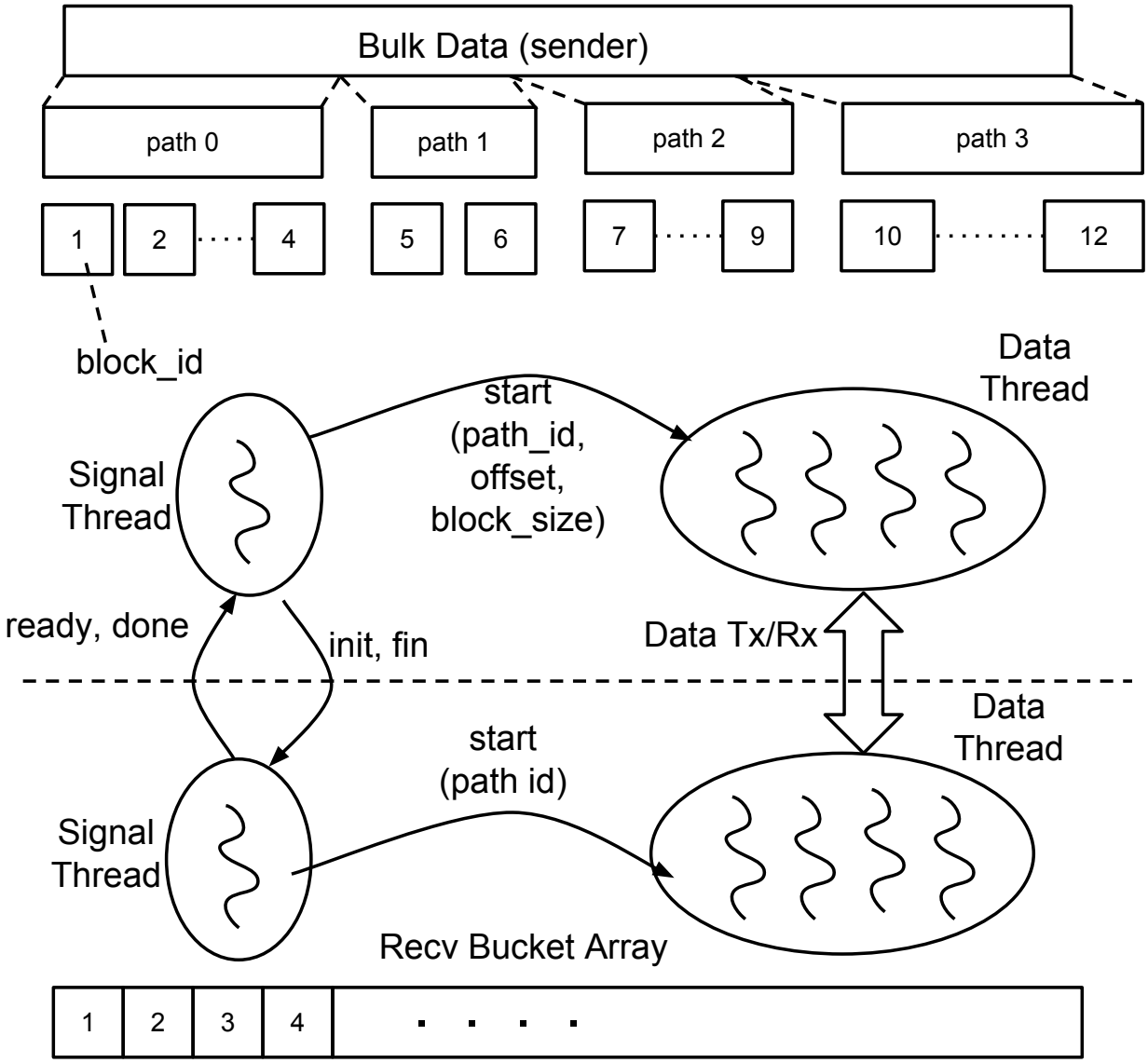

Figure 4-8. A Design pattern for multi-path data transfer.

sends init along with how many blocks are to be sent over which path and waits for ready. Then, the receiver prepares a bucket array and data threads of the same number of block counts, after which it notifies the sender with ready. Then, the data threads start sending upon the receipt of ready signal. Each block is appended with 8 byte header of which consists of 4 byte length of block_id field and the size of the block. Since each data thread knows how many bytes are to be transferred through the respective socket, it terminates once the received byte is equal to the size of the block. Then the received block data is stored in the

bucket array. Each cloud path and the default Internet path contains a single TCP socket pair between sender and receiver. At receiving side, upon the completion of data, done signal is sent along with chunk_id and path_id. Upon the recipients of done, sender queues next block_id upon the available path. Note that all the signaling and data transfer is performed on TCP connections that guarantee reliable transmission. It is better to establish signal socket on default Internet path rather than cloud path, since an Internet path is more resilient upon failure. In this proof-of-concept implementation of this approach for the experiments conducted in this experiment, failure handling for a case where a cloud node fails during transfer is not implemented. An approach that can be used in this case is to time out and retransmit on another available path to cope with this case.

The purpose of presenting this design pattern is to demonstrate that an application-level protocol can exploit the aggregate bandwidth from multi-path multi-streaming to deliver improved aggregate throughput for data transfers. The core principle of above design is inspired by the GridFTP protocol, which conducts application level striping across multiple TCP streams. Moreover, reordering at block level (across TCP socket) should be carefully designed to minimize computing and memory resources. Currently, this implementation is not optimized to address this issue, as it concatenates blocks at the final phase of the session.

4.8 Evaluation

For the performance evaluation, a prototype has been developed which implements the data transfer pattern of the previous section at the application layer, using multiple TCP streams, for the following reasons. TCP is implemented in the kernel layer of the O/S, making it is difficult to modify the code. Furthermore, given the prevalent existence of middleboxes in the Internet, modifying TCP can significantly hinder applicability [69]. For example, SCTP [84] has extensively been used between telecoms, is implemented in most O/S, and is proven to perform well with multi-path for bandwidth increase; however, it has been not used with end-users simply because most users are behind home routers [9] that embed classical 5-tuple NAT [74] and do not support SCTP. Thus, it is decided to use TCP as is and present a

design pattern in the application layer which exploits multi-path for the improved aggregate throughput.

The design pattern discussed in section 4.7 have been implemented as a pseudo-socket API, where "write()" and "read()" methods are implemented as member functions. While an actual socket-level API implementation can be conceived to leverage the cloud multi-path, it is not within the scope of this research; the scope of the implementation is limited to demonstrate empirically the improvement of throughput using cloud multi-paths. For the same reason, instead of implementing "bind()", "listen()" and "connect()" as the APIs, the end point transport address of cloud paths are given as configurable parameters. In the implementation, the application takes bulk data as a file, then opens the socket and sends the data by calling "write()". At receiving end, the socket is called through "read()" and waits for the completion of data transfer. We implemented a simple application which sends 1 GByte of data using this pseudo socket API.

The transfer completion times in the various paths on our testbed are shown in Figure 4-9. It is observed that a multi-path can guarantee at least the throughput that could be achieved using a single path. Moreover, in some cases, throughput using multi-path can be higher, by a significant factor, than a single path. This happens when one of the cloud paths happen to have as much as or much higher throughput than the default Internet path.

Figure 4-10 shows the selective paths that achieved most throughput improvement (left axis, red bar) by using cloud paths, along with their incurred cost (right axis, blue bar). This data reflects the current cost incurred on Amazon EC2 instances ($0.09 per GB) used in the experiment. Note that the cost is with respect to a 1GB data transfer; larger data transfers would lead to proportionally larger costs, e.g. $90 for a 1TB transfer in the case of "wisc-dall", a scenario where all data are transferred over cloud path. The results show that significant improvements in end-to-end data transfer can be achieved by using cloud-routed paths in some scenarios, if a user is willing to pay the additional costs associated with cloud data transfer.

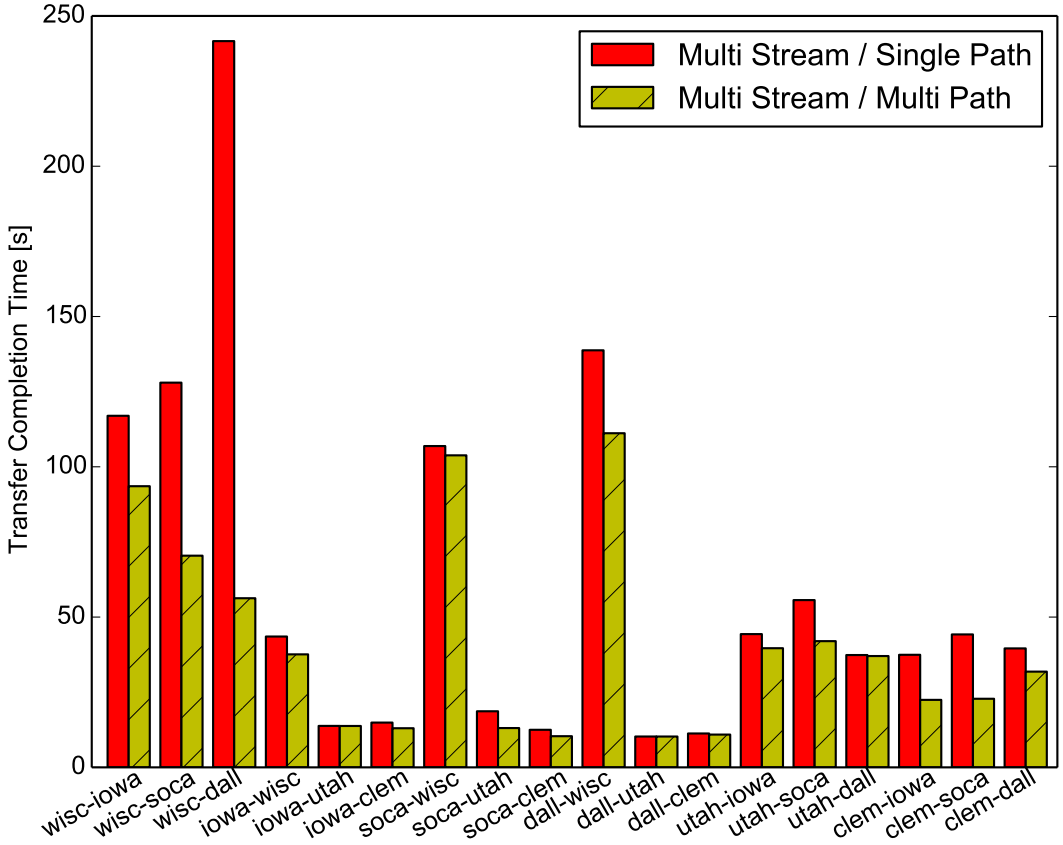

Figure 4-9. Transfer completion time of 1GB data/file in various paths.

The total cost tends to be dominated by data transfer costs — not instance costs — allowing users to trade-off cost for performance depending on service-level objectives.

4.9 Related Work

The GridFTP framework [2] has been widely used across the globe for reliable and secure file transfer. It has extended FTP [72] to incorporate striped and interleaved sending through parallel TCP streams. Originally, FTP uses two TCP connections: control and data. The control connection exchanges commands between the server and the client, while the data connection is created each time a file transfer is necessary. GridFTP extended the protocol to enable parallel TCP by allowing opening multiple data connections and leveraging parallel transfers.

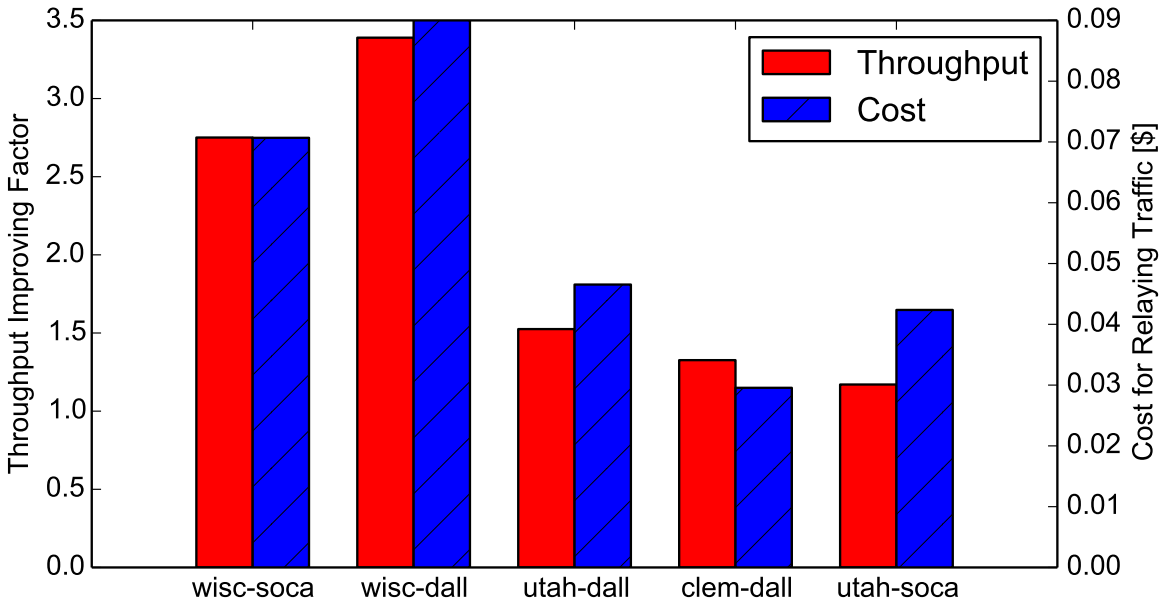

Figure 4-10. Selective path throughput improvement with incurred cost.

Opening multiple TCP sockets and striping data over different paths at the application layer has been studied in related work [82, 32, 33, 34]. In parallel TCP, each TCP stream guarantees in-order delivery of packets, but out-of-order delivery of blocks across multiple TCP streams can stall applications in GridFTP. The approach of Hacker et al. [34] could reduce this reordering by allocating packets on sockets proportional to the size of each congestion window of each TCP socket. However, they also make note that congestion window as a basis for the scheduler which throttles packets to each stream incurs a vicious cycle of either overallocating or starving of socket.

Another drawback of parallel TCP over multiple paths is that it cannot dynamically adapt over the change of the quality of paths [98]. mTCP [98] is a modification from TCP which stripes packets into sub-TCP, where each sub-TCP runs its own congestion control. It considers not only the throughput improvement but also the fairness for public WAN by detecting/avoiding common congestion among paths. Since it is dynamically scoring each path based on the congestion window and outstanding packets, it could adapt over changes of path

characteristics. Moreover, it uses heuristics to find the optimum combination of paths that leads to most independence among paths.

MPTCP [74] is the recent and noteworthy study about implementing transport layer protocol leveraging multi-path. It is implemented to maximize the applicability in current Internet architecture where middleboxes and home routers are prevalent. MPTCP strictly targets to perform no worse than the best single path, and in some cases, it performs worse than the regular TCP. It is implemented in the kernel and is open to the community, and related work has considered it in various platforms and scenarios [54, 11, 67]. Since its target is not maximizing aggregate throughput, in our experience increase in aggregate throughput using MPTCP is not observed in our testbed, an issue also raised in related work [11, 49]. However, in scenarios where there is minimal competing traffic and complete independence of each path [60, 67], MPTCP could improve the aggregate throughput.

In section 4.3, we pointed out that finding the optimal overlay path from the end points is prohibitively expensive. Typical overlays use ping and traceroute or other probing techniques to attain underlay network traits [98] in order to build more suitable virtual topologies: MST algorithm to build multicast tree, nearest peer for P2P file sharing and faraway peer for data replication. However, it has been pointed out that probing to find disjoint paths is not scalable, also redundant as different kinds of overlays duplicate this effort [3, 59]. Nakao et al. [59] suggested a shared routing underlay that overlay can query to attain the underlay network topologies, which could reduce the probing from $O(N^2)$ to $O(N)$ when trying to find disjoint paths among overlay nodes.

4.10 Chapter Conclusion

In Chapter 4, it is demonstrated empirically that cloud instances can be used as a relay point to diversify routing paths between a given source and destination pair. Various experiments show that additional cloud paths can be used to increase aggregate throughput. Especially, most least proximal to the default Internet path happen to provide more additional throughput. However, additional cloud paths do not lead to deterministic throughput increase,

as can be expected with a private line lease; rather, the throughput increase is provided in opportunistic and granular manner, but at a fraction of the cost. Since there was not transport protocol that maximizes aggregate throughput using multi-path at the time of writing this dissertation, instead, an approach inspired from applications such as GridFTP is used, which stripes data across parallel TCP sockets, then pseudo socket APIs is implemented to demonstrate the aggregate bandwidth using multiple paths can lead to actual increase in throughput in the application layer.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, Aug. 2008.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The globus striped gridftp framework and server. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, pages 54–, Washington, DC, USA, 2005. IEEE Computer Society.

[3] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 131–145, New York, NY, USA, 2001. ACM.

[4] L. Andersson and T. Madsen. Provider provisioned virtual private network (vpn) terminology. RFC 4026, RFC Editor, March 2005.

[5] Apache Libcloud. https://libcloud.apache.org/.

[6] Apache Libcloud. https://osrg.github.io/ryu/.

[7] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani. Data center network virtualization: A survey. *IEEE Communications Surveys Tutorials*, 15(2):909–928, Second 2013.

[8] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: Design and implementation of nested virtualization. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 423–436, Berkeley, CA, USA, 2010. USENIX Association.

[9] R. W. Bickhart. Transparent tcp-to-sctp translation shim layer. In *Master's Thesis, University of Delaware*, 2005.

[10] N. Bitar. Multi-tenant data center and cloud networking evolution. In *2013 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*, pages 1–3, March 2013.

[11] C. X. Cai, F. Le, X. Sun, G. G. Xie, H. Jamjoom, and R. H. Campbell. Cronets: Cloud-routed overlay networks. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 67–77, June 2016.

[12] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues. RFC 3234, RFC Editor, February 2002. http://www.rfc-editor.org/rfc/rfc3234.txt.

[13] Chameleon Cloud. https://www.chameleoncloud.org/.

[14] C. Chen, C. Liu, P. Liu, B. T. Loo, and L. Ding. A scalable multi-datacenter layer-2 network architecture. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 8:1–8:12, New York, NY, USA, 2015. ACM.

[15] Cisco. Cisco's Massively Scalable Data Center.

[16] CloudLab. http://cloudlab.us/.

[17] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17 – 35, 2015.

[18] D. Eastlake, A. Banerjee, D. Dutt, R. Perlman, and A. Ghanwani. Transparent interconnection of lots of links (trill) use of is-is. RFC 6326, RFC Editor, July 2011.

[19] H. Eriksson. Mbone: The multicast backbone. *Commun. ACM*, 37(8):54–60, Aug. 1994.

[20] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The locator/id separation protocol (lisp). RFC 6830, RFC Editor, January 2013. http://www.rfc-editor.org/rfc/rfc6830.txt.

[21] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172, March 2015.

[22] A. Fishman, M. Rapoport, E. Budilovsky, and I. Eidus. HVX: Virtualizing the cloud. In *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, San Jose, CA, 2013. USENIX.

[23] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. C. Diot. Packet-level traffic measurements from the sprint ip backbone. *IEEE Network*, 17(6):6–16, Nov 2003.

[24] P. Francis. http://www.icir.org/yoid/.

[25] V. Fuller and T. Li. Classless inter-domain routing (cidr): The internet address assignment and aggregation plan. BCP 122, RFC Editor, August 2006. http://www.rfc-editor.org/rfc/rfc4632.txt.

[26] P. Garg and Y. Wang. Nvgre: Network virtualization using generic routing encapsulation. RFC 7637, RFC Editor, September 2015.

[27] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39(4):51–62, Aug. 2009.

[28] J. Gross and W. Lambeth. Method and apparatus for stateless transport layer tunneling, Sept. 2 2014. US Patent 8,825,900.

[29] N. Grozev and R. Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 44(3):369–390, 2014.

[30] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 15:1–15:12, New York, NY, USA, 2010. ACM.

[31] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, Mar 2001.

[32] T. J. Hacker, B. D. Athey, and B. Noble. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, April 2002.

[33] T. J. Hacker, B. D. Noble, and B. D. Athey. Improving throughput and maintaining fairness using parallel tcp. In *IEEE INFOCOM 2004*, volume 4, pages 2480–2489 vol.4, March 2004.

[34] T. J. Hacker, B. D. Noble, and B. D. Athey. Adaptive data block scheduling for parallel tcp streams. In *HPDC-14. Proceedings. 14th IEEE International Symposium on High Performance Distributed Computing, 2005.*, pages 265–275, July 2005.

[35] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou. Software-defined networking (sdn): Layers and architecture terminology. RFC 7426, RFC Editor, January 2015. http://www.rfc-editor.org/rfc/rfc7426.txt.

[36] S. Hanks, T. Li, D. Farinacci, and P. Traina. Generic routing encapsulation (gre). RFC 1701, RFC Editor, October 1994.

[37] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an autonomous system (as). BCP 6, RFC Editor, March 1996.

[38] E. Hernandez-Valencia, S. Izzo, and B. Polonsky. How will nfv/sdn transform service provider opex? *IEEE Network*, 29(3):60–67, May 2015.

[39] C. Huang, C. Nakasan, K. Ichikawa, and H. Iida. A multipath controller for accelerating gridftp transfer over sdn. In *2015 IEEE 11th International Conference on e-Science*, pages 439–447, Aug 2015.

[40] K. Ichikawa, P. U-Chupala, C. Huang, C. Nakasan, T.-L. Liu, J.-Y. Chang, L.-C. Ku, W.-F. Tsai, J. Haga, H. Yamanaka, E. Kawai, Y. Kido, S. Date, S. Shimojo, P. Papadopoulos, M. Tsugawa, M. Collins, K. Jeong, R. Figueiredo, and J. Fortes. Pragma-ent: An international sdn testbed for cyberinfrastructure in the pacific rim. *Concurrency and Computation: Practice and Experience*, 29(13):e4138–n/a, 2017. e4138 cpe.4138.

[41] J. R. Iyengar, P. D. Amer, and R. Stewart. Concurrent multipath transfer using sctp multihoming over independent end-to-end paths. *IEEE/ACM Transactions on Networking*, 14(5):951–964, Oct 2006.

[42] E. J. Jackson, M. Walls, A. Panda, J. Pettit, B. Pfaff, J. Rajahalme, T. Koponen, and S. Shenker. Softflow: A middlebox architecture for open vswitch. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 15–28, Denver, CO, 2016. USENIX Association.

[43] R. Jain and S. Paul. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11):24–31, November 2013.

[44] V. Jalaparti, I. Bliznets, S. Kandula, B. Lucier, and I. Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 73–86, New York, NY, USA, 2016. ACM.

[45] K. Jeong and R. Figueiredo. Self-configuring software-defined overlay bypass for seamless inter- and intra-cloud virtual networking. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '16, pages 153–164, New York, NY, USA, 2016. ACM.

[46] X. Jiang and D. Xu. *VIOLIN: Virtual Internetworking on Overlay Infrastructure*, pages 937–946. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[47] P. S. Juste, K. Jeong, H. Eom, C. Baker, and R. Figueiredo. Tincan: User-defined p2p virtual network overlays for ad-hoc collaboration. *EAI Endorsed Transactions on Collaborative Computing*, 14(2), 10 2014.

[48] S. Kent and K. Seo. Security architecture for the internet protocol. RFC 4301, RFC Editor, December 2005. http://www.rfc-editor.org/rfc/rfc4301.txt.

[49] R. Khalili, N. Gast, M. Popovic, and J. Y. L. Boudec. Mptcp is not pareto-optimal: Performance issues and a possible solution. *IEEE/ACM Transactions on Networking*, 21(5):1651–1665, Oct 2013.

[50] H. Khosravi and T. Anderson. Requirements for separation of ip control and forwarding. RFC 3654, RFC Editor, November 2003.

[51] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang. Network virtualization in multi-tenant datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, Seattle, WA, 2014. USENIX Association.

[52] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 282–297, New York, NY, USA, 2003. ACM.

[53] S. Krishnan, N. Montavont, E. Njedjou, S. Veerepalli, and A. Yegin. Link-layer event notifications for detecting network attachments. RFC 4957, RFC Editor, August 2007.

[54] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and R. J. Gibbens. How green is multipath tcp for mobile devices? In *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges*, AllThingsCellular '14, pages 3–8, New York, NY, USA, 2014. ACM.

[55] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. RFC 7348, RFC Editor, August 2014. http://www.rfc-editor.org/rfc/rfc7348.txt.

[56] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf. Netagg: Using middleboxes for application-specific on-path aggregation in data centres. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 249–262, New York, NY, USA, 2014. ACM.

[57] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[58] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary. Netlord: A scalable multi-tenant network architecture for virtualized datacenters. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 62–73, New York, NY, USA, 2011. ACM.

[59] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 11–18, New York, NY, USA, 2003. ACM.

[60] C. Nakasan, K. Ichikawa, H. Iida, and P. Uthayopas. A simple multipath openflow controller using topology-based algorithm for multipath tcp. *Concurrency and Computation: Practice and Experience*, 29(13):e4134–n/a, 2017. e4134 cpe.4134.

[61] T. Narten, E. Gray, D. Black, L. Fang, L. Kreeger, and M. Napierala. Problem statement: Overlays for network virtualization. RFC 7364, RFC Editor, October 2014.

[62] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor discovery for ip version 6 (ipv6). RFC 4861, RFC Editor, September 2007. http://www.rfc-editor.org/rfc/rfc4861.txt.

[63] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and M. F. Magalhães. Virtual routers as a service: The routeflow approach leveraging software-defined networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, CFI '11, pages 34–37, New York, NY, USA, 2011. ACM.

[64] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 199–212, New York, NY, USA, 2015. ACM.

[65] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, Aug. 2009.

[66] A. M. Oprescu and T. Kielmann. Bag-of-tasks scheduling under budget constraints. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 351–359, Nov 2010.

[67] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. Experimental evaluation of multipath tcp schedulers. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14, pages 27–32, New York, NY, USA, 2014. ACM.

[68] K. Pagiamtzis and A. Sheikholeslami. Content-addressable memory (cam) circuits and architectures: a tutorial and survey. *IEEE Journal of Solid-State Circuits*, 41(3):712–727, March 2006.

[69] G. Papastergiou, G. Fairhurst, D. Ros, A. Brunstrom, K. J. Grinnemo, P. Hurtig, N. Khademi, M. Txen, M. Welzl, D. Damjanovic, and S. Mangiante. De-ossifying the internet transport layer: A survey and future perspectives. *IEEE Communications Surveys Tutorials*, 19(1):619–639, Firstquarter 2017.

[70] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, 2015. USENIX Association.

[71] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[72] J. Postel. File transfer protocol specification. RFC 765, RFC Editor, June 1980.

[73] L. Qiu, Y. Zhang, and S. Keshav. On individual and aggregate tcp performance. In *Proceedings. Seventh International Conference on Network Protocols*, pages 203–212, Oct 1999.

[74] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? designing and implementing a deployable multipath TCP. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 399–412, San Jose, CA, 2012. USENIX Association.

[75] K. Razavi, A. Ion, G. Tato, K. Jeong, R. Figueiredo, G. Pierre, and T. Kielmann. Kangaroo: A tenant-centric software-defined cloud infrastructure. In *2015 IEEE International Conference on Cloud Engineering*, pages 106–115, March 2015.

[76] L. Rizzo. netmap: A novel framework for fast packet i/o. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 101–112, Boston, MA, 2012. USENIX Association.

[77] R. M. Robinson and P. A. S. Ward. An architecture for reliable encapsulation endpoints using commodity hardware. In *2011 IEEE 30th International Symposium on Reliable Distributed Systems*, pages 183–192, Oct 2011.

[78] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. RFC 3031, RFC Editor, January 2001. http://www.rfc-editor.org/rfc/rfc3031.txt.

[79] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. Stun - simple traversal of user datagram protocol (udp) through network address translators (nats). RFC 3489, RFC Editor, March 2003. http://www.rfc-editor.org/rfc/rfc3489.txt.

[80] T. Saad, B. Alawieh, H. T. Mouftah, and S. Gulder. Tunneling techniques for end-to-end vpns: generic deployment in an optical testbed environment. *IEEE Communications Magazine*, 44(5):124–132, May 2006.

[81] W. Simpson. Ip in ip tunneling. RFC 1853, RFC Editor, October 1995.

[82] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 38–38, Nov 2000.

[83] P. Srisuresh and M. Holdrege. Ip network address translator (nat) terminology and considerations. RFC 2663, RFC Editor, August 1999. http://www.rfc-editor.org/rfc/rfc2663.txt.

[84] R. Stewart. Stream control transmission protocol. RFC 4960, RFC Editor, September 2007. http://www.rfc-editor.org/rfc/rfc4960.txt.

[85] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[86] A. I. Sundararaj and P. A. Dinda. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd Conference on Virtual Machine Research And Technology Symposium - Volume 3*, VM'04, pages 14–14, Berkeley, CA, USA, 2004. USENIX Association.

[87] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, Nov 1997.

[88] J. Touch. Dynamic internet overlay deployment and management using the x-bone. In *Proceedings 2000 International Conference on Network Protocols*, pages 59–68, 2000.

[89] M. Tsugawa and J. A. B. Fortes. A virtual network (vine) architecture for grid computing. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 10 pp.–, April 2006.

[90] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith. Intel virtualization technology. *Computer*, 38(5):48–56, May 2005.

[91] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, March 2010.

[92] M. Westerlund and C. Perkins. Iana registry for interactive connectivity establishment (ice) options. RFC 6336, RFC Editor, July 2011.

[93] D. Williams, H. Jamjoom, Z. Jiang, and H. Weatherspoon. VirtualWires for Live Migrating Virtual Networks across Clouds. In *IBM RESEARCH*. IBM Research Report RC25378, 2013.

[94] D. Williams, H. Jamjoom, and H. Weatherspoon. The xen-blanket: Virtualize once, run everywhere. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 113–126, New York, NY, USA, 2012. ACM.

[95] L. Xia, Z. Cui, J. R. Lange, Y. Tang, P. A. Dinda, and P. G. Bridges. Vnet/p: Bridging the cloud and high performance computing through fast overlay networking. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 259–270, New York, NY, USA, 2012. ACM.

[96] E. Yildirim, D. Yin, and T. Kosar. Balancing tcp buffer vs parallel streams in application level throughput optimization. In *Proceedings of the Second International Workshop on Data-aware Distributed Computing*, DADC '09, pages 21–30, New York, NY, USA, 2009. ACM.

[97] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 203–216, New York, NY, USA, 2011. ACM.

[98] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '04, pages 8–8, Berkeley, CA, USA, 2004. USENIX Association.

BIOGRAPHICAL SKETCH

Kyuho Jeong was born and grew up in Seoul, South Korea. He graduated Sillim High School and attended Hongik University in 2001. He served for Republic of Korea Army from 2003 to 2005, then moved to Australia to practice English and for travel. He returned to the college and obtained his Bachelor of Science degree in electronic and electrical engineering from Hongik University, Seoul, South Korea, in 2008. After graduate, he worked in Samsung Electronics as a logic design and verification engineer. After the industry experience, he was yearning for the study of computer engineering abroad. He received the Master of Science and doctorate degrees in electrical and computer engineering from University of Florida, in 2012 and 2017, respectively. During his PhD study, he focused on virtual network and cloud computing, and published multiple research papers.