Graduate School of Information Science and Technology
THE UNIVERSITY OF TOKYO

Ph.D Thesis
博士論文

Improving Packet Transport in Virtual Networking by Encapsulation Techniques
(仮想ネットワークにおけるカプセル化技術を用いたパケット転送性能の向上)

Ryo NAKAMURA
中村 遼

Abstract

The commoditization of Internet technologies has caused both positive and negative effects in the practical implementation of the Internet. The Internet Protocol (IP), a core protocol of the Internet, has achieved worldwide data communication, however, we cannot modify or replace IP, although it is not sufficient for current demands and requirements. The Internet has become a critical common infrastructure due to its remarkable scalability, technical and operational features. As a result of the deployment and adoption of the Internet technologies, IP has become the integral communication protocol. Almost all of applications have utilized IP for their data communication. Additionally, commoditization of IP often forces us to use commodity IP-based network devices for practical network construction because of their cost per performance considerations. Therefore, it is not feasible today to modify this existing dominant protocol. We have to continue to use IP as is without any modification. Meanwhile, it may be recognized that IP is insufficient for current demands and requirements, which have been diversified by the increase of applications' and users' diversity. For example, virtual machine-based clouds require separations of network domains between users, and the packet forwarding based on IP sometimes causes inefficient link utilization due to shortest path forwarding. Namely, we are facing a dilemma that we have to continue to use IP as is in practical ways, even though we aim to improve IP networking because it is not sufficient for all the requirements from emerging applications.

In this dissertation, we solve the dilemma by focusing on tunneling techniques because tunneling allows us to add new functionalities such as network multiplexing without any modifications to existing protocols and network devices. However, tunneling causes performance degradation at end hosts due to additional protocol processing, and it is not aimed at inefficient link utilization at networks. To tackle the drawbacks of the tunneling approach, we introduce a new architectural view on tunnel-based virtual networking and propose exploiting its potential for optimization. This view separates two aspects of IP networking: host identifiers for data communication and locators for packet transport. Virtual networks that flow through the inside of tunnels are responsible for the identifier space of data communication, and physical networks where encapsulated packets transferred are responsible for simple packet transport. In contrast to the current tunneling design that handles both types of networks by a single network protocol stack, we separate virtual and physical networks into individual network protocol stacks. The network protocol stack for the locator aspect is isolated from the data communication context. Therefore, this separation brings a potential for optimizing the physical network protocol stack to improve packet transport preserving data communication in virtual networks.

Based on the architectural view, we propose two implementation methods that exploit the potential for optimization through changing behaviors of protocol stacks. The first is a new lookup method to avoid the performance degradation due to tunneling. This method improves packet transmission performance at end hosts through optimizing physical network protocol processing separated from end-to-end data communications. The evaluation result of the method shows that the time required to transmit a packet at the end host network stack is reduced, and it improves transmission throughput in five protocols' implementation. By eliminating the additional protocol processing, we improve the performance of packet transport for virtual networking at the end host side. The second is an explicit path control method via a novel usage of tunneling protocols in commodity-based IP data center networks. Data center networks require many network devices to contain many server machines. Hence, using low-end commodity products is an important matter for the data center network construction from the aspect of economic cost.

This method achieves host-driven path control at a physical network by an optimized network protocol stack separated from end-to-end data communications. In this method, end hosts add multiple locators as outer headers to packets, and the packets are routed through specified paths represented by the outer headers. By this path control method, we achieve efficient link utilization and improve the performance of a network. The evaluation result in the case of two data center network topologies shows that the method can utilize multiple paths efficiently with only commodity devices. Through both methods and their evaluation, we demonstrate that exploiting the potential of the architecture can improve network performances without making any modifications to existing protocols and network devices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet has reached nearly around the world. This single data communication network has become a critical part of our society and daily life. Many applications now utilize the Internet as their communication network; consequently, Internet traffic growth knows no boundaries. The growth of the Internet depends on some noteworthy characteristics of the Internet Protocol (IP). The first key point of IP is its robustness. The IP network was originally designed as a distributed packet switching communication network (in contrast to centralized or decentralized networks [5]). A distributed network without centralized nodes can automatically and autonomously tolerate node failures by performing data path recovery that employs alternative nodes and links. Therefore, the distributed network model provides high robustness and fault tolerance. The next key point of IP is its scalability. Scalability is provided by the hierarchical models that are apparent throughout in IP networks: exterior and interior routing architectures, addressing, the network layering model, and so on. Furthermore, interoperability also contributes to the growth of the Internet. The standardized processes used in Internet technologies are open and transparent. Because of a belief in the standards organization, *Rough consensus and running code* [6], and interoperability between different systems, Internet technologies have spread widely across a large number of vendors, products and users.

As a result of the dominance of the Internet and its commoditization, two effects have emerged. The first is that IP has become an integral component of data communication of applications. IP is a general-purpose network protocol; therefore, the majority of recent networked systems, web, P2P, video streaming, database, and so on utilize IP addresses as identifiers and as the end points of their communications. Non-IP networks and communication are relegated to specialized, dedicated purposes such as high-performance computing. The second effect is that IP-based network devices have also become integral components for current network construction from a cost perspective. IP and IP-based network devices have become commodities available at low cost. Consequently, other network systems have changed to IP-based networks even though many of these traditional systems are non-IP. Furthermore, this transition has also promoted the commoditization of IP.

Hence, we are confronted today with the legacy problems of IP as negative side effects of this mass commoditization. IP does not allow us any extensions although IP cannot meet requirements of emerged environments, especially in cloud and data center networks. In virtual machine-based clouds, multiplexing different networks in a physical network is an essential functionality. Each user tenant including traffic, address space and control scope should be isolated. However, IP is not sufficiently able to satisfy this requirement because it was designed for a single scalable network. Another problem is that the forwarding principle of IP sometimes causes a performance problem: inefficient link utilization. The IP routing and forwarding schemes used by standardized commodity network devices obey the shortest path directive; therefore, links excluded from shortest path trees are not used for traffic delivery. Routing protocols running on intermediate nodes construct routing table entries in accordance with shortest path trees from themselves to all destinations, and packet forwarding engines route and forward packets based on those table entries. Therefore, it is difficult to increase the aggregate throughput of a network by simply adding nodes and links because traffic will continue to flow through those links that compose shortest path trees. This restriction prevents data center networks from expanding the aggregate throughput of their networks because such large-scale networks are composed of a large amount of low-end commodity network devices following the shortest path directive.

Although these problems of IP are known, we cannot discard, modify nor replace IP. We have to continue to use IP *as is* without any modification because it is the most fundamental and commoditized component of current data communications. However, there have been a few approaches to mitigate the limitations of IP's extensibility and continue to use the current IP. One is hardware modification and another is the use of tunneling protocols. Carrier backbone networks achieve efficient link utilization through using non-shortest paths and links by sophisticated high-end routers equipped with modified hardware for non-IP based packet forwarding. However, it has still not been achieved in data center networks at a reasonable cost because such networks require a huge number of network devices and servers, compared to the carrier backbone networks. Thus, ideally, data center networks should be constructed from commodity low-end devices that have minimum IP routing and forwarding functions. On the one hand, Tunneling protocols can add new functionality to IP networking without modifications to existing protocols and network device hardware. Tunneling protocols encapsulate a complete packet in a network protocol header again. Network devices route and forward encapsulated packets in accordance with outer network protocol headers such as IP headers, therefore, hardware modification to network devices are not required. Moreover, when encapsulating a packet, a header of a tunneling protocol is inserted between the original packet and an outer header. Tunneling protocols achieve multiplexing different networks through some sort of network identifiers in these tunnel headers.

1.1 Our Position

In this dissertation, we focus on the tunneling approach to improve the extensibility of IP networking without any modifications of IP. Adopting the tunnel approach makes it possible to continue to use existing IP-related protocols and network devices without any modification. However, today's tunneling approaches still have problems for real-world use: performance degradation at end hosts and inefficient link utilization at networks. Tunneling can use IP and existing transport and data link protocols for both inner and outer headers, so that it does not require modifications to the protocols and network devices. However, encapsulation involves additional protocol processing for outer headers in addition to original inner packets. Hence, the processing time required to transmit a packet increase, and the packet transmission performance at end hosts degrades. Furthermore, tunneling protocols are end-to-end protocols. They do not touch packet forwarding behavior at networks. Thus, the inefficient link utilization problem due to the shortest path directive of low-end commodity IP network devices still remains.

To overcome these drawbacks of the tunneling approach, we introduce a new architectural view of tunnel-based virtual networking. Then we propose exploiting this architectural view for optimizing networking performance and link utilization. In the proposed view, tunneling is not yet another virtual link but a boundary between virtual and physical networks. We first illustrate this through analysis of tunneling protocols. Next, we advocate that virtual and physical networks should be separated into individual network protocol stacks unlike today's design, where a single network protocol stack serves both virtual and physical networks. Virtual networks that flow through the insides of the tunnels are responsible for the identifier space of end-to-end data communications. Physical networks are responsible for simple packet transport for the encapsulated packets. Then, the tunnel acts as the boundary between the virtual and physical networks. This separation isolates the locator aspects from data communication and changes the interpretation of physical networks to simple packet transport. Complex packet handling features to manage data communications such as firewall and access control are not needed for physical networks. Namely, the network protocol stack for physical networks can be optimized preserving end-to-end data communications in virtual networks.

Some might think that this architectural view is already known. Some implementations accidentally take the form of the separation; they have different network protocol stacks for virtual and physical networks. However, they do not exploit this potential for optimization. This dissertation reveals that this view has optimization potential to be exploited to overcome the drawbacks of the tunneling protocols. For example, network protocol stacks on virtual machines and hypervisors are different protocol stack instances. Nevertheless, hypervisors and their network protocol stacks are not aware of that they handle packets on physical networks as simple packet transport. In other words, there are excessive functions and features for simple packet transport. It is due to the current design of network protocol stacks on host operating systems. The current design does not have the view for optimization, so that a single network protocol stack implementation deals with both virtual and physical networks. Therefore, existing implementations do not exploit the potential that the physical network protocol stack can be optimized.

As implementation methods exploiting the view's optimization potential, this dissertation proposes two concrete methods. The first method, called overlay FIB, is an optimized packet transmission method for the physical network protocol stack based on the separation. Each layer includes tables and lookups as functions of the layer's protocol, namely IP table lookup and ARP table lookup. The overlay FIB is a new lookup method and a transmission path design that compress multiple lookups on physical networks into one lookup at the tunnel. It eliminates the additional protocol processing typically caused by tunnel encapsulation and, consequently, improves packet transmission throughput. Our evaluation of the overlay FIB implementation shows that it improves packet transmission throughput through tunnels: the throughput of the Linux kernel network processing is approximately doubled in particular protocols.

The second method is an explicit path control method in commodity-based data center networks by an optimized physical network protocol stack design. This method enables end hosts to control paths across a network and split traffic into the paths via the novel usage of tunneling based on the separation. In contrast to the overlay FIB that improves performance at the end host side, this method, called iplb, improves the performance of a network through efficient link utilization. Key ideas of iplb are identifying paths using multiple tunnel headers and shifting path state management from networks to end hosts. iplb uses tunneling protocols supported by the commodity hardware typically found in intermediate nodes to move packets through specified paths. Furthermore, end hosts manage the path states rather than network devices. Thus, network devices do not have to be capable of such path control functions. This requires no router modifications; network operators can build physical networks with commodity IP network devices. We evaluated the method with two types of data center topologies in simulation environments. The results of evaluations on both topologies show that our method achieves better aggregate throughput than typical shortest path forwarding and equal cost multipath.

By introducing an architectural view and proposing exploiting its potential for optimization, we aim to solve the problem of improving IP networks without any modifications to IP itself. The view separates virtual and physical networks. IP addresses are still identifiers for data communication in virtual networks, and commodity IP devices can be used to construct scalable physical networks. Moreover, our proposed methods demonstrate that exploiting the potential can overcome the drawbacks of tunneling. The overlay FIB eliminates the performance degradation due to tunnel encapsulation at the end host side. iplb achieves efficient link utilization in data center networks using only commodity IP devices. Overall, we improve packet transport in virtual networking while leaving current network protocols and equipment unchanged.

1.2 Contributions

The contributions of this dissertation are as follows:

1. **Introducing a new architectural view of virtual networking**
   Based on an analysis of tunneling protocols, we introduce a new architectural view of virtual networking and propose exploiting its potential for optimization. This view separates the identifier aspect for data communication and the locator aspect for simple packet transport into individual network protocol stacks. Isolating the locator aspect from data communication enables us to optimize the network protocol stack for locators to improve packet transport without making any changes to existing applications, network protocols, and network devices.

2. **Proposing a lookup and encapsulation method for tunneling**
   We propose a new destination lookup and encapsulation method for tunneling as a demonstration of exploiting the potential. This method improves the packet transmission throughput of end hosts by optimizing the network protocol stack for locators. This method saves us from performance degradation due to the increase of protocol processing for virtual networking.

3. **Proposing an explicit path control method in commodity-based networks**
   We further propose an explicit path control method in commodity-based layer-3 data center networks as another demonstration of exploiting the potential. This method achieves multipathing and efficient link utilization through an optimized network protocol stack for path control.

1.3 Organization

The remainder of this thesis is organized as follows. The next chapter summarizes the Internet architecture, with emphasis on scalability and commoditization and describes the problems inherent to the current network architecture. Chapter 3 presents an analysis of tunneling protocols and introduces our architectural view of tunnel-based virtual networking and its potential for optimization. Chapter 4 and 5 propose two concrete methods that exploit the potential for optimization. Chapter 4 describes a first method, which overcomes performance degradation due to tunneling at the end host side. Chapter 5 describes another method to achieve efficient link utilization using only commodity network devices. Finally, Chapter 6 concludes the dissertation.

Chapter 2

Internet Architecture and Its Problems

The massive scale deployment of the Internet has led that the networking technologies used in Internet applications and devices have become commodities: consequently, it is not practical to discard or modify such fundamental networking technologies, even if they are not sufficient for emerging requirements. This chapter first summarizes the Internet technologies from the viewpoint of scalability and commoditization. Next, we discuss problems due to its commoditization and existing techniques to mitigate the problems.

2.1 A Worldwide Packet Switching Network

The Internet is one of the most successful data communication networks. One reason for the success factor of the Internet is its scalability. The Internet now covers the entire plane planet. This scalability is offered through following notable design principles.

1. The Internet is a distributed packet switching network.
2. The Internet uses a hierarchical model.
3. The Internet is built and works with the End-to-End principle.

In contrast to circuit-based networks such as the telephone network, the Internet is based on a distributed communication network [5] that uses packet switching. In a packet switching network, data is divided into packets (originally called *message blocks*) and routed and forwarded through intermediate nodes. Packet switching transport enables network resources such as links to be shared during a given time interval (time-sharing system). Furthermore, distributed networks have no centralized nodes. When formerly available links or nodes are broken, routing tables are updated and, subsequently, packets are routed through alternative links and nodes, automatically and autonomously. Distributed network concept also includes inter-networking. A network is composed of multiple networks, in which gateways connecting multiple networks exchange routing information and packets between different networks. These concepts are the important original ideas behind the Internet.

Inter-networking is still a key concept of the Internet to achieve scalability. From a high-level viewpoint, the Internet is composed of Autonomous Systems (AS) as shown at the left side of Figure 2.1. AS is a unit of an organization or a campus scale network composed of multiple gateways and sub-networks. Each AS is operated by an individual operational domain. The ASes are connected to each other and exchange routing information and traffic using exterior gateway protocols. AS-level failures are autonomously recovered by AS-level route convergence using the exterior gateway protocols. Each AS network is also a distributed communication network. Gateways, called routers, interconnect multiple sub-networks as shown at the right side of Figure 2.1. Intra-AS-level failures such as router or link breakdowns recover by finding alternative paths and updating the routing

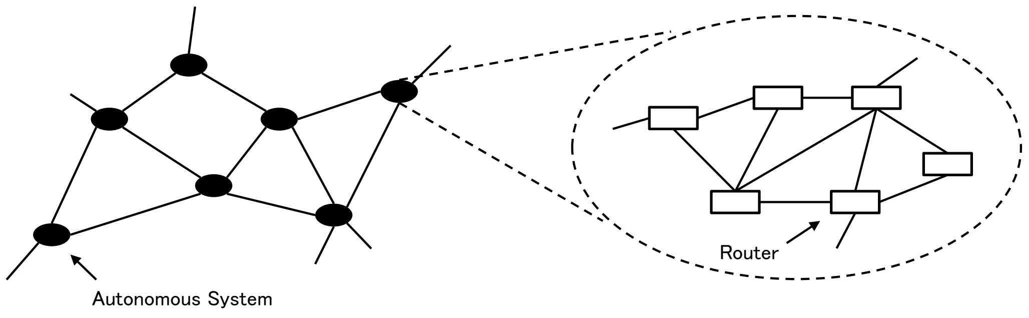


Fig. 2.1. The distributed network model of the Internet. The left side depicts an AS-level distributed network and the right side depicts an intra-AS distributed network.

tables in the routers. Hence, the distributed network concept used throughout the routing levels of the Internet offers high robustness and scalability.

A hierarchical model is also an important feature of the Internet for scalability and simplicity. Hierarchical models provide simplicity through shielding details of lower-level components from higher-levels. Internet routing architecture is hierarchically separated into inter-AS and intra-AS levels as shown in Figure 2.1. The key of hierarchical separation lies in the structure of the Internet Protocol (IP) address. The IP version 4 address [7] is a 32-bit identifier for hosts, routers and networks. The IP address encodes two portions: a host number and a network number (prefix). The network number is encoded into higher bits and the host number is encoded into the lower bits of the IP address. The bit boundary of host and network numbers is indicated by a netmask or prefix length. This netmask method enables network prefixes to be aggregated. For example, in the address 10.0.0.0/16, the network prefix is 10.0.0.0 and the netmask is the highest 16 bits, covering all the longer prefixes from 10.0.0.0 to 10.0.255.255. IP addresses are allocated by the Internet Assigned Numbers Authority (IANA) as address blocks (ordinary /8 networks) to five Regional Internet Registries that manage further finer allocations to National Internet Registries (NIR). The NIRs further allocate longer prefixes to the ASes in their countries. This IP address allocation and aggregation structure is called Classless Inter-domain Routing (CIDR) [8].

CIDR contributes to scalability by compressing routing table sizes of routers in inter-AS routing. CIDR enables prefixes to be aggregated in higher-level routing operations, and ASes exchange aggregated prefixes that are shorter than the /24 prefixes in Inter-AS routing. This aggregation acts to compresses and reduce the number of forwarding table entries required for the router hardware in global-scale networks. Today, the number of IPv4 routing table entries of the Internet is approximately 600,000 prefixes. This number is still growing; however, forwarding table compression methods have been proposed [9, 10].

Another important hierarchical model is the network layering model based on the Open System Interconnection (OSI) reference model. In order to reduce design complexity, this layering model was introduced to protocol design for end-to-end data communication over networks. Each layer contains service, protocol and interface. Services are primitive operations provided by each layer. The layer n communicates with a peer on layer n of the destination host using layer $n - 1$ services. Data containing control information such as a layer n destination address are delivered from layer n to $n - 1$ repeatedly and, finally, transferred through the lowest layer 1, which is physical media. Control information is transmitted as a header that precedes transmitted data. Protocols are concrete procedures used between communication pairs on each layer. In a sense, a protocol is an implementation of the services available in a layer. Thus, layer n protocols exist for each

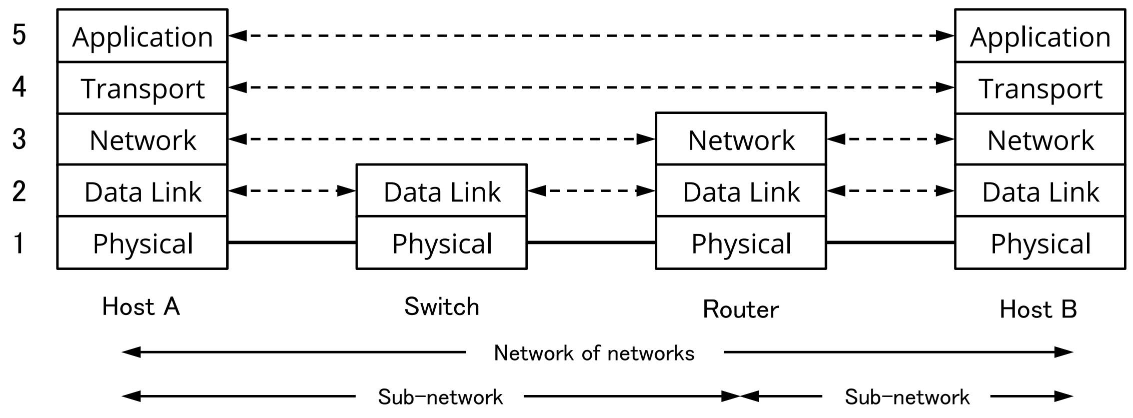


Fig. 2.2. The network layering model introduced by Andrew S. Tanenbaum, from *Computer Networks* [1].

layer. Interfaces are boundaries between layers. Interfaces define how to exchange data across the adjacent layers and primitive operations of the layer. The interfaces of layer n are separated from layer n protocols; therefore, when a protocol of layer n changes, layer $n+1$ does not need to consider the change as long as the interface of layer n is invariant.

The OSI reference model includes seven layers: Application, Presentation, Session, Transport, Network, Datalink and Physical layers. However, it does not represent practical networks such as the Internet [11], thus, we adopt a network layering model introduced by Andrew S. Tanenbaum [1]. The layering model shown in Figure 2.2 comprises five layers. Each layer has corresponding services and its own protocols.

1. **Physical:** This layer represents actual physical links. This layer delivers bits over wired or wireless access media. Popular examples are optical fibers, twisted pair cables, twinax and WiFi.

2. **Data Link:** This layer delivers packets across one hop over a shared network (sub-network). Link-level error corrections, collision detection and congestion control are roles frequently assigned to this layer. For example, Ethernet is a data link layer protocol.

3. **Network:** This layer is responsible for delivering packets even across many hops over multiple networks. It delivers packets from arbitrary source end hosts to destination end hosts so that the logical topology between hosts is a full-mesh network. In addition, it deals with route calculation for packet delivery among arbitrary end hosts. IP is the main protocol in this layer.

4. **Transport:** This layer is responsible for end-to-end communications. It converts application data into packets and reconstructs application data from received packets. Guaranteeing end-to-end data transmission reliability by retransmitting packet in the event of packet loss is another role of this layer when required. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are the most-used transport protocols.

5. **Application:** This layer is an actual program instance communicating with peer. Various applications exchange their data using the lower layers as end-to-end communication networks. For example, Web browsers at client hosts receive their contents via Hyper Text Transfer Protocol (HTTP) over TCP.

Functionalities, physical scopes and scales of each layer are carefully designed and separated to achieve scalability. The lowest layer consists of physical links that connect two nodes. The distances of these links range from a few meters to tens of kilometers. The data link layer (layer-2) is responsible for delivering packets across multiple links in a sub-network. Intermediate nodes on this layer are called switch, and their forwarding process is called switching. The identifier scope of switching (e.g., MAC address on Ethernet) is confined to a sub-network. The network layer (layer-3) delivers packets across multiple data link sub-networks known as a network of networks. This layer does not consider the data link layer identifier. Meanwhile, the layer-3 address space (e.g., IP address) is shared among the entirety of connected layer-3 networks. The scalability of layer-3 is also achieved by the hierarchical routing model described earlier. Finally, the transport layer and application layer protocols communicate end-to-end on top of layer-3 networks. Hence, in the layering model, a layer-2 network composed of physical links and a layer-3 network composed of layer-2 networks achieves scalability on a step-by-step basis. The physical scale of networks can be extended in accordance with the layer levels.

A network such as the Internet represented by this layering model is a single-domain network wired into a physical topology. A link is a physical link, and a layer-2 network composed of physical links. In the layer-3 network, the routes consisting of sets of destinations, next-hops and outgoing ports are also wired into the physical network topology [1]. A next-hop always exists at the opposite side of a link. In addition, to communicate between all end hosts connected to a network of networks, the layer-3 identifiers must be unique for all the entities in the network. Thus, a network represented by the layered model is a single physical network that shares a single layer-3 address space.

The end-to-end principle is also a key concept of scalability. This principle made intermediate nodes and networks simpler by shifting complex functionalities such as reliable data delivery to end hosts. Intermediate nodes are responsible only for delivering packets. Reliability requirements for transferred data correctness should be guaranteed by end host error control. This is true on some large and unreliable networks, although implementing retransmission in the networks themselves is sometimes useful as a performance enhancement measure [12]. The Internet follows this principle. Layer-3 routers route and forward packets from input ports to output ports, and layer-2 switches switch Ethernet frames. There is no explicit reliability measure except packet-level and link-level checksumming. Instead, transport layer protocols in the end hosts are responsible for ensuring the correctness and completeness of data transferred between communication pair. This scheme keeps routers and switches simple and let the network scale.

From the viewpoints of routing and the end-to-end principle, the IP address performs two tasks, namely, it acts as both locators for packet routing and identifiers for end-to-end data communication. Layer-3 routing and forwarding operations use the IP address as the locator that indicates networks to which destination hosts connect. Routers exchange network prefixes and next-hops for constructing routing tables and, then, deliver packets in accordance with the destination IP addresses of the packets. The IP address indicates the node location in the IP network. Meanwhile, end hosts use IP addresses as identifiers to identify both themselves and their communication peers in IP networks. In this manner, both the locator aspect and the identifier aspect are encoded into the IP address simultaneously.

2.2 Commoditization of Internet Technologies

Internet technologies have undoubtedly been commoditized due to their scalability, and propagated into many different areas. Transition to commoditization status can be viewed from various aspects. Cisco systems forecast that the expected IP traffic growth rate is 22% from 2015 to 2020 in the Cisco Visual Networking Index [13]. In 2015, total IP traffic was 72.5 exabytes per month, but traffic volume is still growing. Cisco also forecasts that the number of devices connected to IP networks will be three times as high as the global population in 2020. Moreover, people are working to connect everything to the Internet. This trend, called Internet-of-Things (IoT), aims to connect various devices such as sensors, vehicles, furniture, home electrical appliances, and so on to the Internet. Consequently, IP and Internet technologies will become even more commoditized and widespread.

In addition to scalability, another reason for commoditization is standardization processes for Internet technologies. Open standardization and interoperability between devices from different vendors also promote commoditization of IP network devices. Specifications of these technologies are discussed and determined in an open organization, called the Internet Engineering Task Force (IETF), that allows everyone to participate even individuals. Furthermore, interoperability is the most important factor of Internet standards and products. It is expected that products with completely and correctly implemented standardized protocols will cooperate with other products. Through standardization, IP networks can be constructed using different products implemented by different vendors. Network communities around the world further encourage this ecosystem by supporting discussion and interoperability tests.

As the result of this commoditization, IP becomes an integral protocol from viewpoints of applications and network devices. IP is a general-purpose protocol; consequently, a variety of applications utilizes IP networks for data communications. Mobile applications also use IP networks as discussed in the report [13]. In particular, cloud computing has moved ahead using Internet facilities. Data are stored in data centers, and applications running on the user side handle and process data transmitted across the Internet. End hosts identify themselves and their communicating hosts using IP addresses when they use the Internet. Although many communication protocols involve different transport and application layer protocols such as HTTP, Message Queues, they all use IP networks and IP addresses for communication in the distributed environment. Consequently, IP is the essential protocol and the IP address represents the communication endpoint, making IP networks essential for applications.

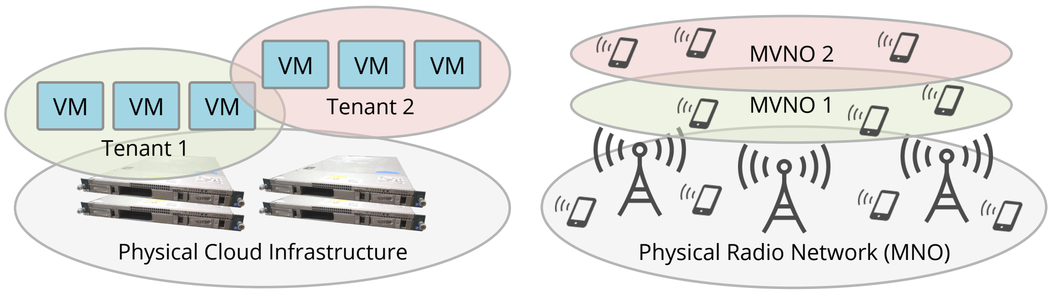
IP-based network devices are also integral to current network construction from a cost perspective. Commodity IP devices such as routers and switches tend to be inexpensive due to the competitive markets fostered by open standards and interoperability. Additionally, the Internet has proved the scalability of IP networks. Hence, non-IP networks are transforming into IP networks. Cellular networks are an obvious example. The early cellular networks like W-CDMA and GSM supported connection-switched networks to provide voice communications. On the other hand, modern cellular networks such LTE integrate all data transport including voice communications into IP-based packet switching networks. Telephone networks are facing the same trend as cellular networks. As announced by NTT in 2015 [14], the Public Switched Telephone Networks in Japan will be migrated to IP networks. IP becomes dominant as locators for packet routing and forwarding by varied networks. On the other hand, non-IP networks are still used for dedicated purposes such as high-performance computing [15, 16, 17]. Networks that do not need to be widely connected can be optimized for dedicated systems. But as a result of commoditization, IP network devices are now dominant; in a sense, most organizations are forced to use IP devices from a cost standpoint.

2.3 Problems

As a result of dominance and commoditization, we have to continue to use IP *as is*, although IP does not meet all the recent requirements. IP lacks extensibility because of its commoditization, so that we have two concrete problems in real use: no domain separation and inefficient link utilization. Applications are diversified; therefore, they involve various demands for networks. Emerging applications, in particular, have various requirements that did not exist when IP was designed. For example, cloud computing, which is based on virtualization technologies, needs a more flexible network design and construction. Moreover, the growth in traffic volume and the importance of services in data centers require high throughput and high availability to data center networks. However, IP does not meet such requirements because IP was originally designed for a single, physically scalable network.

The most significant lacking feature of IP is extensibility. IP does not allow us any extensions to its concrete protocol specification such as header formats. Most proposals of extensions to IP that modify the protocol specification or discard IP have required the modification of all nodes connected to the Internet, applications and network device. Applications running on top of IP networks can be diverse, however, modifying or replacing IP is not so easy. To improve networking by modifying existing network protocols or deploying new protocols, the updated or new protocols must be standardized and implemented in devices, and all devices in the network must be replaced. Such wholesale upgrades and deployments are extremely difficult because current IP networking devices are already deployed and under service operations. The most popular example of the difficulty of deploying a new protocol is IP version 6 (IPv6). IPv6 is a new network layer protocol proposed to overcome limitations of IPv4 in 1995 [18]. The main objective was to expand the number of addresses, thus the number of hosts connectable to the Internet because the number of addresses supported by the 32-bit IPv4 address is insufficient. IPv6 therefore adopted the 128-bit address composed of the 64-bit network address and the 64-bit host address. After 11 years, most major routers and applications offered support for IPv6; nevertheless, the migration to IPv6 has not yet been completed. For example, the percentage of Alexa Top 1000 websites currently reachable via IPv6 is below 20% today [19]. The aggregated IPv6 traffic in the Amsterdam Internet Exchange (a popular Internet Exchange in Europe) is a maximum of approximately 70Gbps compared with the IPv4 maximum, which is over 4Tbps [20]. Accordingly, adopting new network layer protocols is too difficult and may prove to be infeasible even if people around the world make significant efforts. IP networks lack extensibility; however, modifying or replacing IP is not practical. Hence, IP itself is a prime barrier to improve networking.

The first concrete problem is that IP cannot support network domain multiplexing because IP was designed for a single domain and physically scalable network. Nevertheless, domain separation is an important requirement for emerging environments: virtualized environments and data center networks. This domain includes purpose, user, operator, traffic and address space. Networks are sometimes separated in accordance with the domains. As described in Section 2.1, the Internet is also a single layer-3 network composed of multiple sub-networks. The operational domains of each AS can be unique; however, the whole network is still a single layer-3 network from an address space viewpoint. Layer-3 networks that construct a layer-3 network by interconnecting with each other share a common address space; therefore, the layer-3 identifier must be unique at all nodes in the layer-3 networks. Hence, IP address duplication is prohibited in an IP network except in very particular cases such as anycast [21]. Currently various applications and systems



(a) Virtual machine based cloud. (b) Mobile Virtual Network Operator.

Fig. 2.3. Example cases requiring the domain separation in a physical network.

utilize IP networks for their communications; consequently, requirements to separate domains in a physical IP network have emerged.

Figure 2.3 shows two example cases requiring domain separation. The first case involves the Infrastructure as a Service (IaaS) clouds that are widely deployed and used. IaaS maintains physical server and network facilities, and provides Virtual Machines (VMs) running on those physical facilities to customers as depicted in Figure 2.3(a). The IaaS model cloud is a fundamental infrastructure of modern society because people can utilize the virtual server resources without having to individually manage physical facilities. In IaaS clouds, customer domains (tenants) should be separated from each other. Of course, customers are individual operators; therefore, they have different purposes, requirements and system designs. The customers want to use arbitrary numbers of VMs and arbitrary addresses for VMs, and they may intend to construct their own tenant networks, to achieve their individual needs. Virtual server resource isolation is easy because VMs on a hypervisor (HV) run independently from each other. However, multiplexing different network domains in a physical network is not supported by IP.

The Mobile Virtual Network Operator (MVNO) shown in Figure 2.3(b) is another example of the need for domain separation. MVNO is a mobile service provider who does not own wireless network infrastructure including radio stations. Instead, an MVNO provides mobile network connectivity to its customers through wireless network infrastructure owned by Mobile Network Operator (MNO). MNO sells bandwidth to multiple MVNOs. An MVNO's wired infrastructure is usually connected to the Internet and includes systems interconnected to the MNO's network. Customer traffic is exchanged at the interconnecting point between MNO and MVNO. For example, an end-user's cellular phone seems to be served directly by MVNO mobile networks. However, packets transmitted from cellular phones are delivered to destinations through the wireless network infrastructure of MNO to the wired infrastructure of MVNO. Naturally, MNO networks must distinguish between the various MVNO user packets and deliver those packets to the correct MVNO unlike their direct MNO customers. Also in this case, the existence of multiple domains means that MNO and MVNO traffic must be multiplexed onto the single physical MNO network. This means that the current layering model and its actual implementation with plain IP is not well-suited for the MVNO systems, which needs domain separation on a single physical MNO network facility.

The second concrete problem is inefficient link utilization due to the IP's routing mechanism based on the shortest path directive. In IP networks, it is difficult to utilize multiple links to deliver traffic efficiently. Routing algorithms and packet forwarding hardware for IP are based on sink trees and shortest path forwarding. In any topology, the shortest paths from a given source to all destinations form a tree rooted at the source as shown in

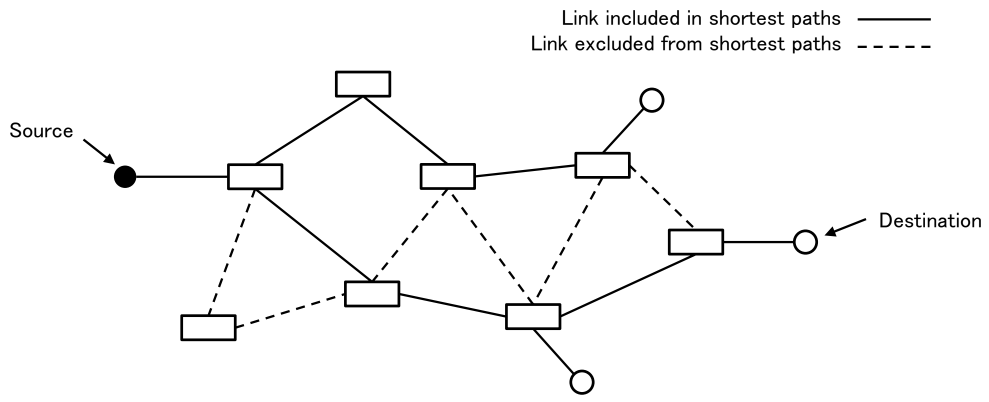

Link included in shortest paths
Link excluded from shortest paths
Source
Destination

Fig. 2.4. Sink tree and shortest path forwarding on an IP network: Links excluded from shortest paths are not used to deliver traffic.

Figure 2.4. Routers route and forward packets along the tree, so that the routing tables of routers are composed of pairs: a destination and a next-hop. The routing and forwarding engines of commodity IP routers are manufactured on this basis. Hence, even if links exist that are excluded from sink trees from all sources in a network, IP routers cannot utilize them to deliver additional traffic. Shortest path forwarding also means that it is not easy to improve the aggregate throughput of a network by adding links and routers because the additional links will not be used unless they form parts of shortest paths.

2.4 Existing Techniques to Mitigate the Limitation

Various techniques that have aimed to mitigate the insufficiencies of IP discussed in the previous section have been proposed, and several have been deployed. The basic approach to the domain separation problem is to add an additional network identifier to a protocol header that identifies the domain. The basic approach to improve network aggregate throughput is to employ multipath routing for packet transport. This section summarizes existing techniques and describes their drawbacks.

2.4.1 Domain Separation

Multiplexing different domain networks in a single physical network requires some sort of identifier embedded into the packet protocol headers to distinguish the networks to which the packets belong. Moreover, network devices must be aware that each packet belongs to a particular network. Devices find next-hops for forwarding packets based on corresponding forwarding tables for different networks, and end hosts work the same way. Thus, protocols having such network identifiers have been proposed.

The Virtual LAN (VLAN), the most popular and practical way to achieve network domain separation, is a technique to add a network identifier to the Ethernet header. The VLAN was first proposed as an Ethernet enhancement in 2003. The VLAN adds a 32-bit field to the original Ethernet header that contains 12-bit tag called a VLAN ID that distinguishes the sub-networks to which Ethernet frames belong. VLAN-aware switches have forwarding tables for each VLAN ID, and tagged Ethernet frames are forwarded in accordance with the corresponding forwarding table. Thus, the VLAN achieves network multiplexing in a physical network by adding the network identifier to the layer-2 protocol header. However, the problem with the VLAN is scalability. VLAN is a layer-2 protocol;

```
+----------------------------------+
| Outer Ethernet Header (14byte)   | \
+----------------------------------+
|     Outer IP Header (20byte)     |  Outer (encapsulation) headers
+----------------------------------+
|       NVGRE Header (8byte)       | /
+----------------------------------+
| Inner Ethernet Header (14byte)   | \
+----------------------------------+
|     Inner IP Header (20byte)     |
+----------------------------------+  Inner (original) Ethernet frame
|                                  |
|         Original payload         |
|                                  | /
+----------------------------------+
```

Fig. 2.5. The NVGRE packet format.

therefore, the scale of networks to which VLANs can be deployed is limited to physical topologies as shown in Figure 2.2. All the switches in a network must be aware of the VLAN. Moreover, intermediate switches must learn destination MAC addresses and outgoing ports to manage forwarding tables. Consequently, the number of hosts in a network requires an identical number of hardware forwarding table entries in switches. In addition, the number of multiplexable networks on a VLAN-based network is limited to 4095 due to the 12-bit VLAN ID. These scalability issues mean that the VLAN is not feasible for current VM-based cloud environments, which must accommodate millions of hosts and thousands of tenant networks within a single data center.

Network virtualization overlays [22] based on the *tunneling* approach are yet another solution for domain separation. In contrast to the layer-2 protocol enhancement, they do not require modifications to existing protocol headers and network devices through encapsulation techniques and tunneling protocol headers including such network identifiers. Tunneling is a technique to connect distant networks across IP networks. Layer-3 tunnel devices in source networks work as routers and encapsulate IP packets in IP headers. Encapsulated IP packets are transferred to the destination network in accordance with the destination IP addresses of the outer IP headers and decapsulated by tunnel devices in the destination networks. Then, the decapsulated original IP packets are routed and forwarded in the destination networks in accordance with the original destinations of the inner IP headers. In contrast to traditional layer-3 tunneling protocols, which encapsulate IP packets in IP headers, in the new tunneling protocols, network virtualization overlay encapsulates Ethernet frames in IP headers. For example, Figure 2.5 depicts the encapsulated Ethernet frame format by Network Virtualization Using Generic Routing Encapsulation (NVGRE) [23], which is one of the network virtualization overlay protocols. The inner Ethernet frames are transferred across IP networks without any modification by IP encapsulation. Such tunneling protocols have tunnel headers that include network identifiers similar to the VLAN ID. The NVGRE header shown in Figure 2.5 has 24-bit Virtual Subnet ID (VSID) to identify the networks to which the inner Ethernet frames belong. These tunneling protocols are commonly used in cloud environments. HVs encapsulate the VM traffic, and VM networks are provided as layer-2 Ethernet networks among different HVs across IP networks.

Network virtualization overlays offer benefits for networks and their operators: scalability and mobility. Tunneled packets seem to intermediate nodes to be simple IP packets; consequently, intermediate networks can be constructed as layer-3 IP-based networks that have better scalability than layer-2 Ethernet-based networks. Further, intermediate IP routers do not need to learn the inner-destination MAC addresses, which helps limit the number of forwarding table entries required. In addition to scalability, tunneling protocols provide separation for the two aspects of the IP address: locator and identifier. The communication identifier is the IP address of the inner IP header, while the locator is the IP address of the outer IP header. Packets to a VM are transferred to an HV that accommodates the VM by using the outer IP header destination as the locator. This locator identifier separation provides mobility. Consequently, VMs can be located at arbitrary HVs without address (identifier) changes. As VMs are migrated among different HVs, only the outer IP addresses (locators) change, not the inner IP addresses (identifiers) of the VMs. For these reasons, network virtualization overlays have been deployed to cloud environments. Many protocols have been proposed such as NVGRE, Virtual eXtensible LAN (VXLAN) [24], Stateless Transport Tunneling Protocol (STT) [25] and Generic Network Virtualization Encapsulation (Geneve) [26].

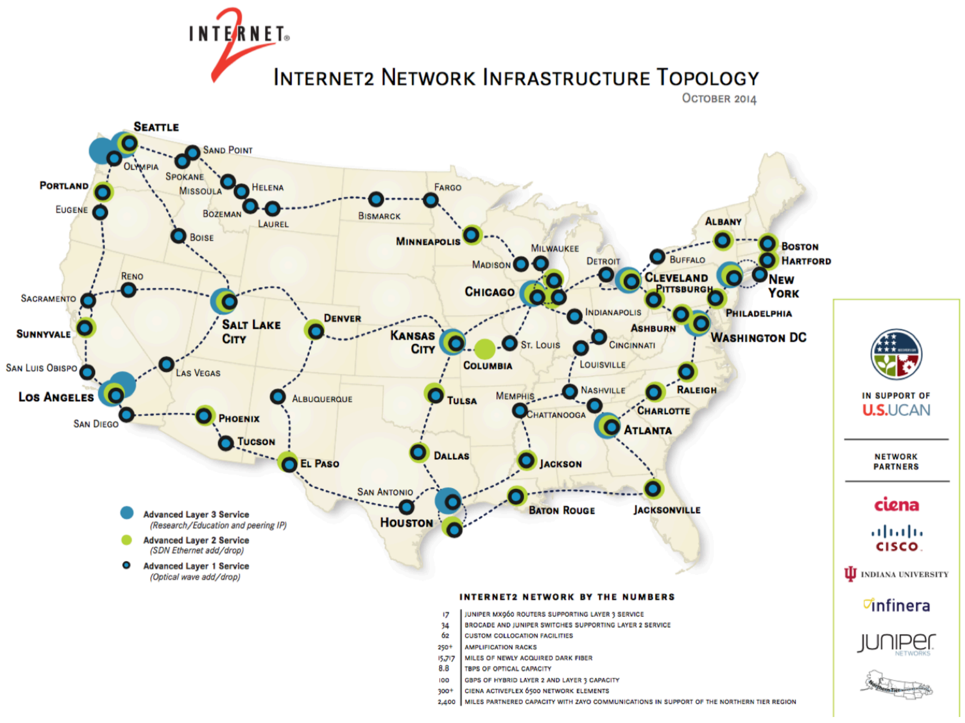
2.4.2 Efficient Link Utilization

The basic approach for efficient link utilization is to use multiple paths to transport data between a sender host and a receiver host by using non-IP based packet forwarding hardware instead of traditional IP routers that are only capable of shortest path forwarding. Multipathing is a packet forwarding technique that packets are routed and forwarded from a source to a destination by intermediate nodes through multiple paths. Consequently, the aggregate throughput of a network is improved by using the extra links, which are excluded from shortest paths. Multipathing techniques have been widely considered. The process of optimizing aggregated throughput in networks is called traffic engineering: a topic that includes how to divide traffic into flows and how to adjust flow and path assignments based on traffic load.

The most popular technique for multipathing and traffic engineering is Multiprotocol Label Switching (MPLS) [27]. Since MPLS is a non-IP based packet forwarding technique, if we adopt this solution, we have to discard IP-based network devices. MPLS encapsulates an IP packet in the MPLS shim header and the outer Ethernet header. IP packets are encapsulated and transferred through an MPLS network, and decapsulated by the egress nodes of the MPLS network. Additionally, Ethernet over MPLS encapsulation also exists. The MPLS shim header has a 20-bit identifier called a label that is used as the key to determine the outgoing ports by MPLS intermediate nodes. MPLS constructs paths, called label switched paths, using combinations of labels and outgoing ports in MPLS nodes. Multiple label switched paths from a single source to a single destination can exist. Namely, MPLS adapts to path construction for multipathing. These MPLS-like techniques are called *path-switching* in which intermediate nodes transfer packets along *paths* as virtual circuits embedded in networks.

Deployments of multipathing into real networks are categorized into two cases based on the types of target networks: carrier backbone networks and data center networks. Networks adopt multipathing for aggregate network throughput improvement tend to be large networks such as Internet Service Providers (ISPs) backbone networks and data center networks such as cloud or Content Service Providers (CSPs). In contrast to enterprise networks such as office branches, these types of networks currently require over 100-Gbps network capacities and transport large amount of customer traffic. Therefore, to reduce the cost of network facilities, operators of such networks aim to make full use of links including non-shortest path links.

The first case is to adopt multipathing in carrier backbone networks of ISPs. This is relatively easy because router products for such large backbone networks usually already support MPLS, and numbers of necessary routers are small. General ISP backbone networks interconnect geographically distributed tens of Points of Presence (PoPs) that serve



INTERNET2 NETWORK BY THE NUMBERS

17	JUNIPER MX960 ROUTERS SUPPORTING LAYER 3 SERVICE
34	BROCADE AND JUNIPER SWITCHES SUPPORTING LAYER 2 SERVICE
62	CUSTOM COLLOCATION FACILITIES
250+	AMPLIFICATION RACKS
15,717	MILES OF NEWLY ACQUIRED DARK FIBER
8.8	TBPS OF OPTICAL CAPACITY
100	GBPS OF HYBRID LAYER 2 AND LAYER 3 CAPACITY
300+	CIENA ACTIVEFLEX 6500 NETWORK ELEMENTS
2,400	MILES PARTNERED CAPACITY WITH ZAYO COMMUNICATIONS IN SUPPORT OF THE NORTHERN TIER REGION

Fig. 2.6. The Internet2 Network Infrastructure Topology in 2014 from Combined Infrastructure Network Map [2]. Non-bordered blue circles indicate layer-3 PoPs, lime green circles indicate layer-2 PoPs and small bordered blue circles indicate layer-1 PoPs.

customers in each area. For example, in 2014, Internet2, a production service network intended for academics research in the United States, had 10 layer-3 PoPs and links between PoPs including layer-2 and layer-1 PoPs as shown in Figure 2.6. Hence, the IP routers for backbone networks require small numbers of high bandwidth ports and high capacity backplanes. Additionally, the number of routers required is less than in data center networks. Therefore, these IP router products, called high-end routers, tend to be expensive and have enhanced functionalities such as MPLS.

In contrast to carrier backbone scenarios, data center networks have different requirements for their IP routers. In data center networks, a large number of inexpensive commodity routers is needed to accommodate a large number of servers. The main task of ISP backbone networks is to connect geographically distributed PoPs. In contrast, the main task of data center networks is to connect thousands or even millions of servers. These servers contain VMs or content, and the data center networks transport VM traffic and content data from servers to the Internet or to other internal servers. The NIC link speed of a server is lower than backbone network links; however, a router or a switch must accommodate as many servers as possible. Furthermore, data center networks require many such routers or switches to accommodate all their servers, interconnected routers and switches. Therefore, in contrast to backbone networks, they cannot afford to use expensive high-end products because so many network devices are needed to construct data center networks. Instead, data center network operators demand commodity low-end routers and switches. The Open Compute Project [28] established by Facebook in 2011 aims to advance the commoditization of server and switch hardware specialized for data center efficiency from the cost aspect using an open source approach and standardization.

Table 2.1. Limitations of existing techniques.

	No modification	End host performance	Network performance
Path-switching for link utilization	✗	-	✓
Tunneling for domain separation	✓	✗	-

Hence, the main requirements of IP routers and switches for data center networks are that they must have a larger number of ports, be commodities and inexpensive products.

Commodity IP routers do not include enhanced functionalities such as MPLS because those techniques are typically limited to backbone routers and high-end products. Therefore, commodity-based data center networks constructed from inexpensive routers that have minimum IP routing and forwarding functionalities cannot achieve efficient link utilization. Many multipathing approaches have been proposed [29, 30, 31, 32, 33] that have not been deployed in real-world environments because they require hardware modifications to network devices to achieve efficient multipathing. Requiring dedicated hardware is a barrier to deployment because it increases costs. On the other hand, there is a commodity-based multipathing technique called Equal-Cost Multipath (ECMP) [34, 35, 36]. ECMP is supported by low-end commodity routers, however, it is known to be inefficient in making full use of extra links [32] because of hash-based traffic balancing (this problem is described in Chapter 5 in detail).

2.5 Summary

As a result of the commoditization and lack of extensibility of IP, we have to continue to use IP as is to continue to use existing applications, protocols and network devices. However, IP is not sufficient to meet a wide spectrum of current demands. It does not support domain separation, and its link utilization is inefficient. Despite these problems, the modification of IP itself and its practical deployment is extremely hard because of lack of extensibility due to the commoditization. In other words, we are facing the problem that despite the fact that IP is insufficient to address the problems of domain separation or link utilization, there is no practical way to discard, modify or replace IP.

Existing techniques to mitigate the limitation of IP are inadequate under the constraint of the non-extensibility of IP and performance perspectives. Table 2.1 summarizes two existing techniques that abandon using IP for efficient link utilization or end host performance for domain separation. We must continue to use IP *as is* to use existing applications and commodity IP-based network devices. Therefore, techniques without modifications to IP are feasible solutions. *No modification* on Table 2.1 means that techniques do not require any modifications to existing IP-related protocols and devices. From this viewpoint of no modification, MPLS based on the path-switching approach discards IP to improve *network performance* through multipathing, so that it is not feasible for such data center networks requiring inexpensive commodity network devices. Meanwhile, network virtualization overlays based on the tunneling approach do not require modifications to IP; however, it causes a degradation of *end host performance* due to additional protocol processing.

As we discussed above, the path-switching approach provides aggregate throughput improvements through multipathing to achieve better network performance; however, this approach requires hardware modifications to intermediate nodes. Non-IP based packet forwarding techniques such as MPLS require high-end and expensive products to achieve multipathing. They are applicable for carrier backbone networks composed of small numbers of high capacity routers to accommodate geographically distributed areas. However, it is not practical in data center environments that require large numbers of low-end network devices. In contrast to carrier backbone networks, data center networks have to accommodate and interconnect large numbers of server machines. Data center operators' need for commodity network devices in constructing their networks makes this approach infeasible from a cost aspect.

Network virtualization overlays based on the tunneling approach achieve domain separation without any modification to existing protocols and network devices, however, they involve performance degradation during packet transmission at end hosts. Tunneling requires additional protocol processing at end hosts compared with original packet transmission. Through the transmission path, data transmitted from applications is divided into packets by a transport layer protocol, and the packets are processed by layer-3 and layer-2 protocols for virtual networks. Then, packets are encapsulated and further processed by layer-3 and layer-2 protocols for lower physical networks. Consequently, the time required to transmit a packet increases at the end hosts, and transmission throughput decreases. In addition, tunneling is an end-to-end technique, so that the problem of inefficient link utilization at networks still remains.

A recent approach to relieve the performance degradation due to tunneling is hardware offloading, however, it is inefficient for various tunneling protocols as known as protocol ossification [37]. TCP Segmentation Offload (TSO) is an offloading technique to reduce the number of packets processed in kernel space. TSO-capable Network Interface Card (NIC) receives larger packets (generally 64KB) than the Maximum Transmission Unit (MTU) size of Ethernet (which is generally 1500-byte) from the end host software network stack. Then, the NIC divides these large packets into MTU sized packets in hardware. This reduces the number of packets that the end host software network stack must handle and, consequently, transmission throughput is improved. TSO is a dedicated technique for TCP packets; TSO-capable NICs cannot handle encapsulated packets even if the inner packets are TCP because the encapsulated packets appear to be UDP or tunnel header. Some tunnel-aware TSO NICs have been published [38, 39]. These NICs can distinguish encapsulated TCP packets and provide TSO functions. However, this hardware offloading approach is not applicable for all tunneling protocols, but is specific for a particular tunneling protocol. Such NICs are aware of only the particular tunneling protocols implemented in their NIC hardware. Therefore, when a new tunneling protocol were introduced and deployed, new NIC hardware, which can recognize this new protocol, must be developed and operators may have to replace all the NICs in their networks. Thus, the recent ad hoc offloading approach is inefficient for various tunneling protocols.

Chapter 3

Virtual Networking

In this chapter, we introduce a new architectural view of tunnel-based virtual networking and propose exploiting its potential for optimization. This view changes interpretations of inner and outer networks of tunnels and brings a potential for optimizing behaviors of protocol stacks to overcome the disadvantages of tunneling. First, this chapter explores details of tunneling protocols. Based on the exploration, we argue that tunneling is a way to achieve isolation of the identifier and locator aspects of the Internet Protocol. Further, we introduce the new view through a network architecture for virtual networking and its potential for optimization.

This dissertation focuses on the tunneling approach to improve IP networking because adopting tunneling makes it possible to continue to use existing IP-related protocols and network devices without any modification. However, its extensibility is inadequate because conventional tunneling causes performance degradation at end hosts due to encapsulation. Moreover, tunneling is an end-to-end technique, so that the inefficient link utilization problem at the network side still remains. Through enhancing tunneling, we aim to achieve domain separation and efficient link utilization with performance improvement at end hosts and networks.

3.1 Details of Tunneling Protocols

The tunneling approach was originally proposed to provide virtual networks. A virtual network is a network constructed from virtual links unbound from physical links. The original IP network is fully wired to physical topologies and is dependent on the wiring of links, switches and routers. Thus, address space and network policies (e.g., security policies) are also tied to physical network structures. Virtual networking has emerged from a demand from people who wish to overcome the inflexibility of physically wired network structures. A Virtual Private Network (VPN) is one example of virtual networking: users connect to distant office networks at which individual network policies are applied through VPN circuits across public networks. The VM-based cloud environments described in Section 2.3 is currently the most important tunnel use case. Here, the customers' networks are fully separated from the physically wired cloud providers' networks. The key technology for achieving virtual networking under the constraint of the non-extensibility of IP is the tunneling approach.

The behavior of tunneling protocols is the virtual link connecting two distant hosts across shared IP networks. Figure 3.1 shows the protocol stacks at end hosts using tunneling. Applications running on both hosts communicate with each other across a virtual link. The IP protocol module used by the application on host A works as if it were directly connected to the IP protocol module on host B via the virtual link. The tunneling protocol module on host A encapsulates an original IP packet into an outer IP header whose

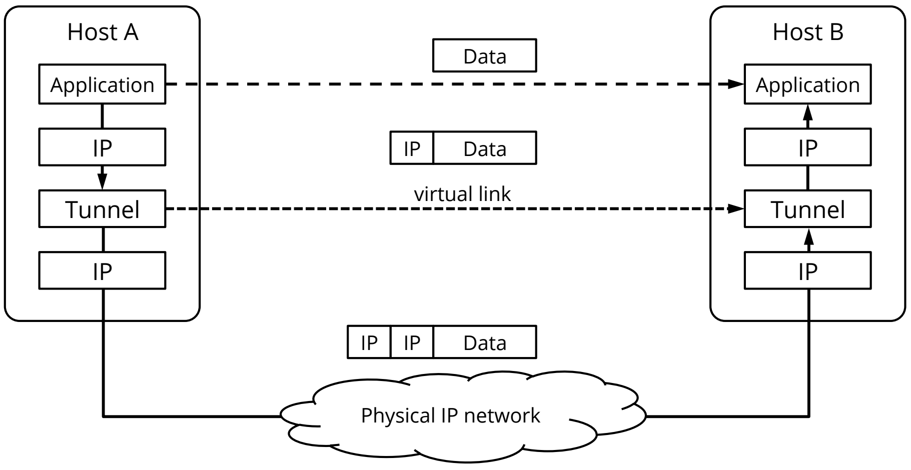


Fig. 3.1. An example of protocol stacks and tunneling communications between two hosts across a physical IP network.

destination address is the address of host B on the physical IP network and then transmits the encapsulated packet to the network. The encapsulated packet is routed and forwarded to host B in accordance with the outer destination address. Then, the tunneling protocol module on host B decapsulates the packet, and the IP protocol stack receives the original IP packet as though it had been received directly from the virtual link. This is the basic tunneling behavior. In VPNs, a virtual link performs as a VPN circuit. End hosts can connect to a distant office network through the virtual links as VPNs that span public IP networks. In VM-based cloud environments, the VMs contain the applications and upper IP protocol modules. VMs placed in different HVs can perform as if directly connected through virtual links even if the HVs are located in geographically distant data centers. Moreover, multiplexing virtual links on a physical IP network between same hosts can be achieved by adding some sort of network identifier that distinguishes the virtual networks of inner packets as described in Section 2.4.1 using NVGRE as an example. Namely, tunnels perform virtual links, and utilizing virtual links for network construction is virtual networking.

3.1.1 IP over IP Protocols

A basic tunneling format is layer-3 over layer-3 tunneling protocols that encapsulate IP packets into IP headers. Tunneling protocols are explained as *layer-i* over *layer-n* protocols. Layer-i indicates the protocol level of the encapsulated inner packet. The layer-n indicates the protocol level of the outer header, which is an actual network protocol for transporting the encapsulated packet. Ordinary tunneling protocols work over layer-3 because they aim to expand networks through virtual links across scalable IP networks. Tunneling first appeared in 1988, RFC1057 Distance Vector Multicast Routing Protocol (DVMRP) [40]. In this original use case, tunneling was used to connect distant multicast-enabled IP networks across multicast-disabled IP networks. RFC1057 states that:

> A tunnel is a method for sending datagrams between routers separated by gateways that do not support multicasting routing. It acts as a virtual network between two routers. For instance, a router running at Stanford, and a router running at BBN might be connected with a tunnel to allow multicast datagrams to traverse the Internet.

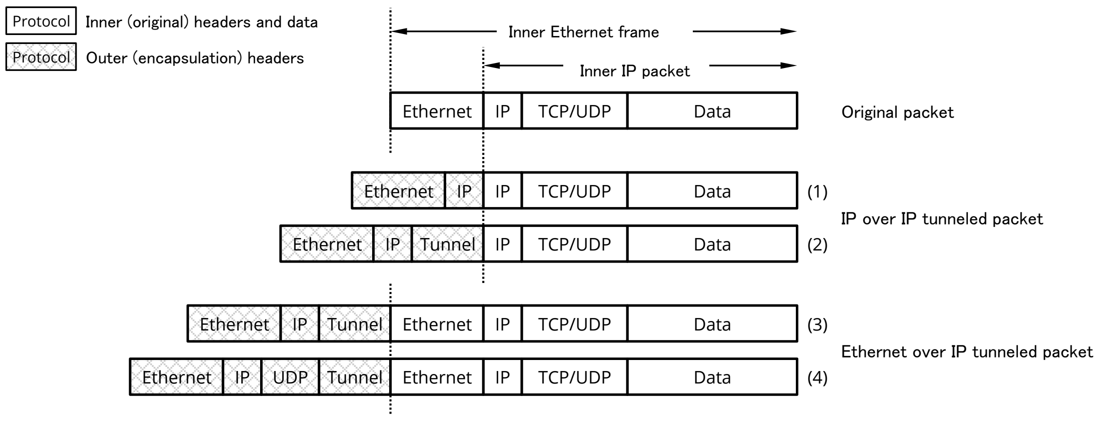

Protocol Inner (original) headers and data
Protocol Outer (encapsulation) headers
Inner Ethernet frame
Inner IP packet
Ethernet
IP
TCP/UDP
Data
Original packet
(1)
(2)
IP over IP tunneled packet
Tunnel
(3)
(4)
UDP
Ethernet over IP tunneled packet

Fig. 3.2. Packet header formats of the original IP packet, IP over IP and Ethernet over IP tunneling protocols.

DVMRP tunnels transfer IP multicast packets across unicast-only IP networks to expand a multicast routing domain. In other words, tunneling was originally introduced to expand a network to which individual policies (multicasting in this case) were applied across public networks. A DVMRP tunnel was implemented using the loose source route option, which an original source address and a destination multicast address were contained in IP option fields, and a source tunnel endpoint address and a destination tunnel endpoint address were inserted to form the proper source and destination address fields of an IP header. Twenty-eight years later, the loose source route option is obsolete because of security issues. Moreover, the Protocol Independent Multicast used as the modern multicast routing protocol makes DVMRP obsolete.

After DVMRP, tunneling was more generalized to involve connecting two distant IP networks. The IP-in-IP encapsulation protocol (IPIP) was proposed and standardized in 1995 [41]. IPIP is a very simple IP over IP tunneling protocol. It encapsulates an IP packet in an IP header as depicted in image (1) of Figure 3.2, so that routers in physical networks route and forward encapsulated packets in accordance with the destination IP addresses of the outer IP headers. Similarly, receivers decapsulate the packets, and the original IP packets are received by the IP protocol module again; delivered to transport protocols or routed and forwarded again in accordance with their inner destination addresses. The packet format of IPIP is shown in image (1) of Figure 3.2, where the outer IP header is next to the original IP header. IPIP was originally defined as IPv4 over IPv4 encapsulation. Additionally, IPv4 over IPv6 and IPv6 over IPv4 tunneling also exist as variants of IPIP. Generic Routing Encapsulation (GRE) is also a popular IP over IP tunneling protocol [42]. In contrast to IPIP, GRE adds the GRE tunnel header between the inner IP header and the outer IP header as depicted in image (2) of Figure 3.2. The GRE header contains a version number, the inner protocol type and checksum fields. In addition, it has optional key and sequence number fields. An important characteristic of IPIP and GRE is that they are point-to-point tunneling protocols. The virtual links of IPIP tunnels and GRE tunnels assume that the links connect a source node and a destination node statically configured in advance. In other words, a virtual link seems to be a point-to-point link to the inner IP protocol modules. In Figure 3.1, the tunnel protocol module of IPIP or GRE always encapsulates all packets from the inner IP protocol module to the identical destination host.

Locator Identifier Separation Protocol (LISP) [43] is another IP over IP tunneling protocol; however, it is not point-to-point tunneling. LISP introduces a routing mechanism to tunneling to achieve mobility. LISP was designed to separate the locator and identifier aspects of the IP address. The LISP tunnel protocol contains a table, called a map table, composed of pairs of endpoint identifiers (EIDs) and corresponding routing locators (RLOCs). EIDs are identifiers of nodes in the inner virtual networks: the IP addresses on virtual IP networks and, RLOCs are locators of nodes on outer physical networks: IP addresses on physical IP networks. By using this map table rather than a static configuration consisting of a single source and destination pair for a tunnel, LISP achieves multipoint tunneling and can be dynamically updated. When the LISP tunnel protocol module receives an IP packet from an upper IP protocol module, it searches the map table for the appropriate RLOC (the destination IP address on physical networks) for the destination IP address of the original packet (EID). After this lookup, the original IP packet is encapsulated in a LISP tunnel header and an outer IP header whose destination IP address is the found RLOC. Through this separation of identifier and locator, LISP provides mobility; if a node location (RLOC) changes, the node identifier (EID) for data communication need not change. Instead, only the locator for the identifier in the map table entry is updated. Therefore, communications such as TCP sessions are not disrupted. Consequently, the tunneling mechanism of LISP differs greatly from point-to-point tunneling protocols, although its encapsulation format is similar to GRE as shown in image (2) of Figure 3.2. To the inner IP protocol module, the LISP tunnel protocol module seems to be a virtual link; however, it works as a multipoint link by looking up the locators on physical networks in accordance with the identifiers on virtual networks.

3.1.2 Ethernet over IP Protocols

Another major tunneling format is layer-2 over layer-3 tunneling protocols that encapsulate Ethernet frames into IP headers. Motivation to encapsulate Ethernet frames into IP is to connect virtual machines at different and distant hypervisors. Network interfaces of virtualized host OSes (Guest OSes) are Ethernet interfaces because this design allows us to use existing Ethernet device drivers for guest OSes and accommodate multiple VMs in the same sub-network. Accommodating VMs in an Ethernet network enable operators to adopt the identical network operation techniques for virtualized environments as is used in physical environments. Therefore, the inner Ethernet header performs solely as a protocol header for compatibility; there is no necessity for Ethernet protocol services such as collision detection and flow control. The Ethernet header is used to define sub-networks and use the existing network protocol stack of guest OSes as is. Another demand is to extend a sub-network across IP networks. Network policies are applied at the sub-network level; consequently, Ethernet over IP tunneling is needed to construct a sub-network to which an identical policy can be applied even to multiple distant branches across shared IP networks. For these reasons, Ethernet over IP protocols are called network virtualization overlays.

NVGRE described in Section 2.4.1 is one such Ethernet over IP protocol. Its packet header format is shown in image (3) of Figure 3.2. A variant of NVGRE, Stateless Transport Tunneling (STT) [25] was proposed in 2012. STT aims to utilize the TSO feature of NIC hardware. Ordinary NICs have TSO, however, they are not aware of encapsulated packets. Thus, STT defines the TCP-like STT header inside the outer IP header that the packet format is that depicted in image (3) in Figure 3.2. STT uses the same IP header protocol number for TCP (the protocol number for TCP is 6); therefore, STT-encapsulated packets pretend to be TCP packets. Consequently, STT can utilize the TSO feature of existing commodity NICs. In contrast to the packet header format of NVGRE and STT, in which the tunnel header is inside the outer IP header, the packet header formats of VXLAN [24] and GENEVE [26] are shown in image (4) in

Figure 3.2. The only difference between (3) and (4) is the inclusion of the outer UDP header. The outer UDP header was added based on two practical operational issues. The first issue is that the protocol number field of the IP header that indicates the next header protocol type is 8-bit value that can accommodate only 256 types. Therefore, adding new protocols as the next protocols of the IP header is very difficult because the protocol number resource is limited. In contrast, encapsulating packets in the UDP header can employ the UDP port number field to indicate the next protocol (tunnel header) type. The second issue involves traffic load balancing. General load balancing features of network devices (e.g., routers) determine next-hops of packets for balancing using a 5-tuple (source IP, destination IP, source port, destination port and the IP protocol number) extracted from the packets. Thus, encapsulated packets without transport headers cannot be balanced efficiently. Consequently, the destination UDP port number is used to indicate the inner protocols and the source UDP port number is used to insert entropy for load balancing. Overall, Ethernet over IP tunneling protocols were designed considering these practical operational issues.

One significant characteristic of Ethernet over IP protocols is that they all include some sort of network identifier within their tunnel headers to achieve network multiplexing. The network identifier indicates the virtual networks to which inner original Ethernet frames belong. Receiver hosts can distinguish the virtual networks for the inner Ethernet frames by checking the network identifiers, so that different virtual networks can be multiplexed over a single physical IP network. The names of such identifiers differ: Virtual Subnet ID (VSID) in NVGRE, Context ID in STT and Virtual Network Identifier (VNI) in VXLAN and GENEVE, but their role is the same. It is now the norm that modern Ethernet over IP tunneling protocols will include such network identifier to achieve network multiplexing on a physical network.

Another characteristic of Ethernet over IP protocols is that they are multipoint-to-multipoint tunneling to emulate shared links, for example, to connect multiple HVs. They use forwarding tables called overlay Forwarding Database (FDB) like the LISP map table. The FDB is composed of pairs of Ethernet MAC addresses and corresponding IP addresses. The MAC addresses correspond to the destination MAC addresses of the inner Ethernet frames, and the IP addresses correspond to the destination IP addresses of the outer IP headers. When a host sends an Ethernet frame through a tunnel, the FDB on a tunnel protocol module is used to determine the IP address of the destination host that accommodates the Ethernet interface that has the destination MAC address of the sending frame. As shown in Figure 3.3, when a VM sends an Ethernet frame to a destination VM, the tunnel protocol module in the HV hosting the source VM determines the address of the HV hosting the destination VM by looking it up in the FDB. Then the tunnel module encapsulates the frame into a tunnel header and an outer IP header. As the result, the VMs can be directly connected via a virtual Ethernet link. Interestingly, this is the same behavior as the locator identifier separation mechanism of LISP. The destination MAC addresses of the inner Ethernet headers perform identifiers in virtual networks, while the outer destination IP addresses are the locators in physical IP networks. VXLAN and STT have a mechanism for learning and updating FDB entries that uses IP multicast.

3.1.3 Other Tunneling Protocols

In addition to IP over IP and Ethernet over IP tunneling protocols, there are some tunneling protocols that do not explicitly define inner and outer protocols. For example, MPLS can transport IPv4 or IPv6 packets and Ethernet frames within MPLS headers. Similarly, the Network Service Header (NSH) [44] also does not define outer and inner protocols. A commonality of MPLS and NSH is that both protocols determine the outer

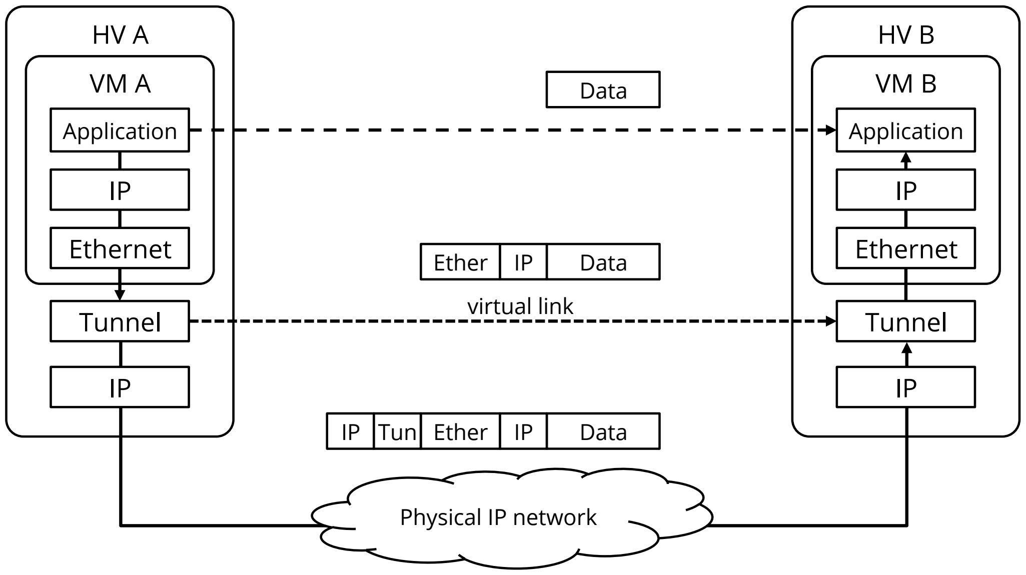


Fig. 3.3. An example of protocol stacks and communication of Ethernet over IP tunneling between two VMs on two HVs through a virtual link.

protocol destination from some sort of forwarding table. MPLS uses the label as the key of forwarding table. The MPLS protocol module determines the outgoing port and the next MPLS node's MAC addresses based on the label table. NSH has similar label fields called Service Path Index (SPI) and Service Index (SI) in the NSH header. The NSH mapping table contains the SPI and SI as keys and the destination host IP addresses and encapsulation formats, respectively, as values. The NSH tunnel protocol module determines the destination host addresses and encapsulation formats in accordance with the NSH mapping table when sending packets. Both, MPLS and NSH also have lookup mechanisms to determine the destination locators corresponding to the destination identifiers.

3.1.4 Layer Violation due to Tunneling Protocols

As a result of this flood of tunneling protocols explored in previous sections, the abstraction of network layering model is violated for practical issues. For example, the Ethernet header appears in protocol stacks twice, but the inner Ethernet protocol does not perform full data link layer services; instead, it is only used as a protocol header for VM network interface compatibility. The UDP header is used for encapsulation due to practical and operational issues but not due to the need for transport layer services. STT pretends to be TCP on the packet level so it can utilize the TSO features of NIC hardware. This disguise violates the transport layer because network devices that are not aware of STT handle STT packets as TCP packets. In addition, tunnel-aware TSO NICs are implemented for specific tunneling protocols [38, 39]; consequently, they are unsuitable for other or new tunneling protocols. As we discussed, the current tunneling protocols are excessively practice-oriented and lack comprehensive views.

3.2 Locator and Identifier

Even though tunneling protocols are different from each other depending on practical issues, the tunneling approach separates two different aspects of the IP address: identifiers and locators, into two IP address fields of their protocol header formats. The identifier

Table 3.1. Lookup mechanisms of major tunneling protocols

Protocol	**Identifier**	**Locator**
IPIP GRE	none	(always single) outer IP address
LISP	inner IP address (Edge ID)	outer IP address (RLOC)
VXLAN NVGRE STT GENEVE	inner MAC address	outer IP address
NSH	Service Path Index and Service Index	outer IP address

aspect is that IP addresses identify end hosts and their communication peers. End hosts establish communication channels using their own and peers' IP addresses as identifiers. The locator aspect is that IP addresses indicate locations of networks to which end hosts connect. Routers deliver packets across networks in accordance with the destination IP addresses of the packets. In usual IP networks without tunneling, IP addresses perform both aspects simultaneously. By contrast, in tunneling, different IP addresses perform both aspects respectively. IP addresses of inner original headers perform the identifier aspect, and IP addresses of outer encapsulation headers perform the locator aspect. Tunneling protocols determine locators in accordance with identifiers of peers of end-to-end communications.

Tunneling that separates two aspects can be regarded as a boundary between virtual and physical networks for identifier and locator aspects of IP addresses. Inner packets flowing through tunnels seem to be accommodated in virtual networks. Applications establish their data communication channels using IP addresses as host identifiers on virtual networks. These IP addresses identifying communicating hosts are contained within inner IP headers of original packets. Packets flowing through virtual links of virtual networks are encapsulated by tunneling protocols, and the encapsulated packets are transferred across physical IP networks using IP addresses as destination host locators. These IP addresses distinguishing locations of communicating hosts are contained within outer IP headers of encapsulated packets. This behavior is independent of concrete tunneling protocols and their packet formats. Virtual networks perform as identifier networks for data communications, and physical networks perform as locator networks for encapsulated packet transport. Then, tunnels act as the boundary between them.

The essential functions of tunneling to work as the boundary between virtual and physical networks can be defined as follows:

1. Determining destination locators according to the inner protocol identifiers.
2. Encapsulating packets into outer protocols.

Determining destination means including mechanisms for looking up outer destination IP addresses that are destination host locators. Ethernet over IP protocols such as VXLAN and NVGRE use FDB to find destination IP addresses based on the inner-destination MAC address of Ethernet headers. LISP uses the map table to determine the destination RLOCs according to EIDs. Point-to-point tunneling protocols such as IPIP and GRE do not require such lookup tables because they always have a single tunnel destination. In other words, they maintain and search tables composed of a single entry. Table 3.1 lists the major tunneling protocols and their lookup mechanisms. They all use some sort of inner protocol identifiers as keys for lookups and determine the outer protocol locators on physical networks. After lookups, tunneling protocols encapsulate packets

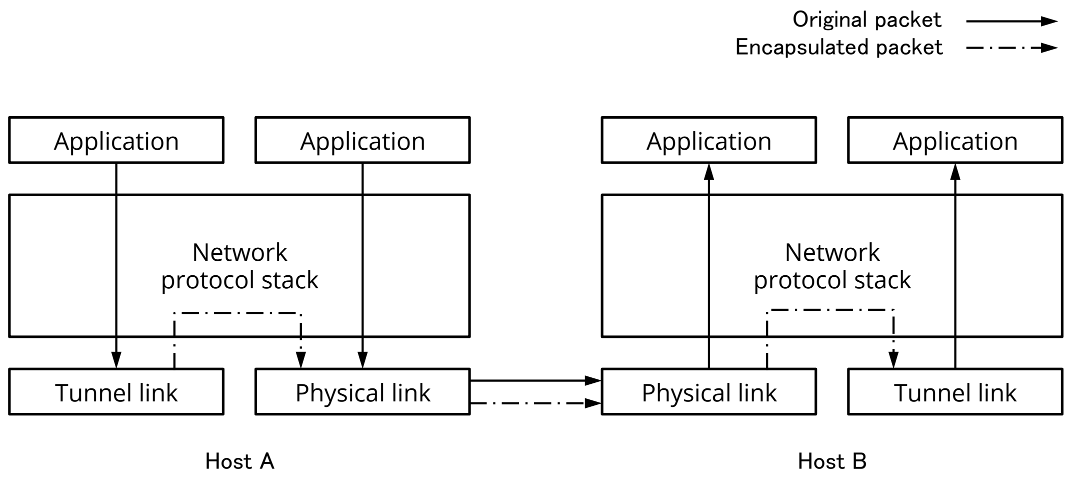


Fig. 3.4. The current network stack design with tunneling: Two hosts connect to each other through a physical link and a virtual tunnel link. The inner network and the outer network share the same network domain simultaneously.

into appropriate outer network protocols. Ordinary, the protocols encapsulate packets into either IPv4 or IPv6 headers. Some protocols such as VXLAN and GENEVE add the UDP header in addition to the IP header for practical reasons. Outer header protocols are determined by their protocol contexts in a manner similar to inner protocols. Hence, the common functions of all tunneling protocols can be abstracted into two functions: determining destination and encapsulation. Tunneling performs at the boundary of virtual networks and physical networks using these two functions.

3.3 Problems of the Current Network Protocol Stack Design

From the viewpoint that tunneling is the boundary between virtual and physical networks, a single network protocol stack currently serves different networks: the virtual network as an identifier space and the physical network as a locator network. Figure 3.4 depicts the design of the current network protocol stack with tunneling. Packets transmitted to a tunnel virtual link are encapsulated in the outer IP headers and pushed back to the same network protocol stack again. Then, the network protocol stack processes IP routing lookup and determines the next-hops of the encapsulated packets in accordance with the outer IP addresses (locators). As shown in Figure 3.4, the current design does not separate the identifier and locator aspects of the IP address because a single network protocol stack handles both the original and encapsulated packets in the same network domain, e.g., address space and routing table, simultaneously.

This design, a single network protocol stack for both types of networks, has advantages for implementation simplicity and problems on additional complexity and performance. The current design is derived from the original use of tunneling, in which a tunnel was intended to be simply a virtual link across shared networks. In other words, this design does not provide different protocols for different types of networks. The design that pushed back encapsulated packets provided some implementation benefits—allowing designers to use the same network protocol stack implementation for both inner and outer packet handling. It also provided modularity for protocol implementations. To add a new tunneling protocol, it was necessary only to implement a new tunnel link device driver. Meanwhile, this design also causes some problems. First, a single network protocol stack for both locator and identifier aspects causes additional complexity. For example, a single routing table for a network protocol stack must have entries for both outer and inner IP packet routing simultaneously. Namely, address spaces of the outer and inner networks are mixed. Second, packet transmission performance will degrade as noted in Section 2.4.1. Packets over tunnels must pass through the same network protocol stack implementation twice; therefore, the time required to transmit a packet increases, and the total transmission performance degrades.

3.4 The Virtual Network Architecture

In this dissertation, we introduce a new architectural view to tunnel-based virtual networking and propose exploiting its potential for optimizing network protocol stacks to overcome the drawbacks of tunneling. As discussed in previous sections, tunneling can be regarded as the boundary between virtual and physical networks that are responsible for identifier and locator aspects. Hence, we advocate that different network protocol stacks across tunnels serve virtual and physical networks respectively because they are different networks. This view brings the potential for optimizing a network protocol stack for the locator aspect preserving end-to-end data communications in another network protocol stack for the identifier aspect.

Figure 3.5 depicts a network architecture, called a virtual network architecture, for tunnel-based virtual networking adopting our view. In this architecture, two different network protocol stacks serve the locator and identifier aspects respectively in contrast to a single network protocol stack of the current design. The virtual network protocol stack is responsible for protocol processing on the virtual network as identifier spaces, and the physical network protocol stack that is isolated from the virtual network is responsible for protocol processing on the physical network for packet transport of encapsulated packets. Tunneling performs a shim layer between physical and virtual networks. Tunneling protocols have mapping tables for identifiers in virtual networks and locators in physical networks, and determine locators of packets in accordance with the tables. To the network protocols of virtual networks, a tunnel appears to be a virtual link. When a packet is transmitted through a tunnel, the tunneling protocol determines the destination on the physical networks according to the packet's destination in the virtual network and encapsulates the packet into a network protocol header compatible with the physical network.

This architecture changes the interpretation of physical networks into simple packet transport by confining end-to-end data communications in virtual networks, so that we can optimize the physical network protocol stack by removing unnecessary functions. Virtual networks are identifier spaces for data communications. Applications establish connections using identifiers and exchange data using transport services on the virtual network protocol stack. Namely, the context of data communication is sealed up in virtual networks. Therefore, network policies are applied to virtual networks. For example, packet-filtering mechanisms (e.g., firewall) are placed and implemented in the virtual network protocol stack because the physical network protocol stack cannot touch the original packets after encapsulation. In contrast, the physical networks are simple packet transport networks without such mechanisms. The addresses of physical networks perform as locators, and the physical network protocol stacks on nodes simply forward packets in accordance with the locators.

Data transmitted from applications is processed by the virtual network protocol stack, a tunneling protocol and the physical network protocol stack at a sender host, and transmitted to a physical wire. Both virtual and physical network protocol stacks are composed

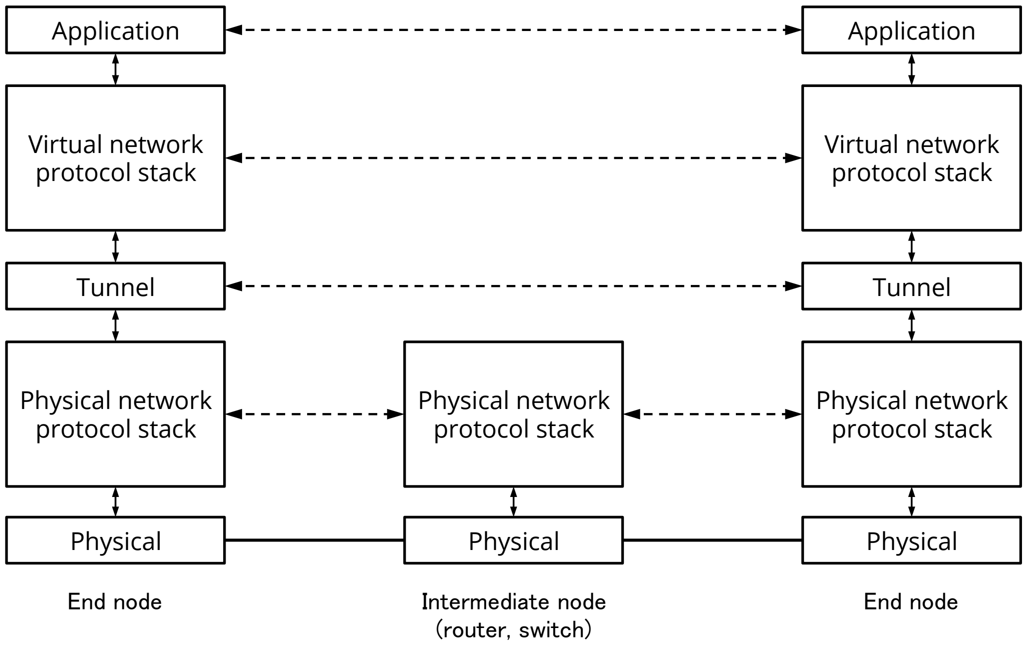

Application
Application
Virtual network protocol stack
Virtual network protocol stack
Tunnel
Tunnel
Physical network protocol stack
Physical network protocol stack
Physical network protocol stack
Physical
Physical
Physical
End node
Intermediate node (router, switch)
End node

Fig. 3.5. The virtual network architecture based on the tunneling approach.

of layer-4, layer-3 and layer-2 protocols as needed. Applications running on hosts communicate with their peer in the same way as in the traditional end-to-end communication model. The only differences from the traditional model are that the data communications of applications are accommodated by the virtual network and its protocol stack: the physical network topologies and locations of hosts are hidden from the virtual network. Consequently, the virtual network protocol stack provides transport services (layer-4) and host identifiers (layer-3) for applications. Applications establish connections using the transport services and the identifiers. Mechanisms achieving network policies are also implemented in and provided by the virtual network protocol stack. Data transmitted by applications are divided into packets, processed by the network protocols of the virtual network, and the packets are finally transmitted to virtual links toward the virtual network protocol stack on the communication peer. The tunnel provides virtual link functions. When a tunnel receives a packet from the virtual network protocol stack, the tunnel determines the destination address of the communication peer on the physical networks: the locator. This determination mechanism works in the same way as the lookup of tunneling protocols. The tunnel searches a mapping table using the identifier of original packets as a key and finds the destination locator corresponding to the identifier. The encapsulated packet is delivered to the physical network protocol stack, transmitted to a wire, and transferred on the physical networks in accordance with the routing mechanisms of the physical networks.

The virtual network architecture utilizes the inheritance of the End-to-End principle [12], so as to avoid modification on intermediate nodes such as routers and switches for practical network construction. In this architecture, end hosts and their network protocol stacks handle concrete virtual and physical network protocol processing, and the intermediate nodes do packet routing and forwarding as is. The intermediate nodes of the physical networks are unaware of the virtual networks and do not care whether the forwarding packets are encapsulated packets. Therefore, there is no need to modify existing network devices and their hardware, and we can continue to use such network devices.

The definition of a tunnel is that it comprises an interface, functions and services like conventional layer definition. The interface of the tunnel as a shim layer facing the virtual network protocol stack is the same as that of physical links. If the bottom protocol of a virtual network protocol stack is Ethernet, a tunnel seems to be an Ethernet link. If the bottom protocol is IP, the tunnel seems to be an IP link. In both cases, Ethernet and IP protocols transmit packets using their protocol headers to tunnels as links regardless of whether the links are virtual or physical. In other words, this abstraction interprets a tunnel to be a network interface. A network interface (which is actually a tunnel) connects to a virtual network. The functions of the tunnel are 1) to determine the destination locators on physical networks according to identifiers on virtual networks, and 2) to encapsulate packets into outer protocol headers on top of the physical network protocol stacks. Therefore, packets on virtual networks are transferred to destination peers through physical networks.

In addition to the interface and functions, the tunnel can provide some services of the shim layer. The most important service is network multiplexing for domain separation. As network virtualization overlays do, tunneling can build multiple virtual networks on a physical network by using network identifiers. When a tunnel receives a packet from a virtual network protocol stack, the tunnel marks the virtual network to which the packet belongs on a network identifier of the tunnel header. The tunnel on the receiver host can then distinguish the virtual network of the received packet, and deliver the original packet to the appropriate virtual network protocol stack. Another considerable service on the shim layer is routing. The logical topology of a virtual network is basically full-mesh. Thus, actual packet transit paths follow shortest paths on physical topologies. However, it is a known issue that using only shortest paths is inefficient from the viewpoints of aggregate throughput and traffic engineering. Therefore, some techniques introducing new tunneling protocols that have routing identifiers at the tunnel layer have been proposed [45, 46]. These approaches construct virtual network topologies separated from physical topologies to optimize networks from the performance view. A service to provide routing on the tunnel layer is called overlay routing.

Our architectural view separates outer and inner networks of tunnels and changes the interpretations of them. The virtual architecture interprets outer networks as simple physical packet transport for encapsulated packets and inner networks as the identifier space for end-to-end data communications. A pair of source and destination IP addresses appears in a packet twice, however, meanings of these IP addresses are different. The IP addresses of inner IP header acts as the identifier for data communication on virtual networks, while the IP addresses of outer IP header acts as the locator for simple packet transport on physical networks. Furthermore, the tunnel acts as a boundary between the virtual and physical networks. Protocol stacks of virtual and physical networks are composed of existing protocols as before. Virtual and physical network protocol stacks handle both networks for identifier and locator aspects respectively.

3.5 Optimization Potential

This dissertation proposes exploiting the potential for optimization derived from the virtual network architecture to overcome the drawbacks of tunneling. We aim to achieve extensibility of tunneling with performance improvement at both end hosts and networks. In addition to original advantages that stem from using tunneling, the physical network protocol stack can be optimized preserving end-to-end data communications by our architecture because physical and virtual networks are separated into individual network protocol stacks. We list these advantages below:

1. Virtual networks are released from physical and geographical topologies.
2. The separation enables us to use existing protocols and implementations in both virtual and physical network protocol stacks.
3. Physical network protocol stacks can be optimized by changing behavior of protocols.

First and second advantages are derived from original tunnel use: mobility and reusability of existing protocols and implementation. Virtual networks are not bound to physical and geographical topologies, which provides mobility: an identifier on a virtual network protocol stack can migrate to other physical network protocol stack instances on other nodes without the disruption of communication. When a virtual network protocol stack is a VM such as Figure 3.3, the VM can move from one HV to other HVs connected to the same virtual network without requiring changes of identifiers [47]. Therefore, VMs can migrate without disrupting data communications. This is one of the major advantages of using network virtualization overlays in IaaS cloud environments. On the other hand, the LISP provides prefix mobility as its one of main benefits. A LISP implementation, LISP-mob [48], provides mobility to Linux nodes. Existing protocols and implementations can be used to achieve both virtual and physical network protocol stacks. This advantage also means that the virtual network architecture does not require any modification to existing protocols. As discussed in Section 2.3, IP itself cannot be modified nor discarded because it is highly commoditized and is now the dominant protocol for data communication. The tunneling approach is highly deployable to existing networked systems despite the fact that IP networks lack extensibility. Adopting the tunnel approach allows us to continue to use existing protocols and implementation.

Based on the architecture, we propose optimizing the physical network protocol stack through changing behaviors of protocols. In the architecture, virtual and physical networks are separated into individual network protocol stacks, so that the physical network protocol stack is responsible for only simple packet transport networks. In other words, the physical network protocol stack does not need to support full networking features. Protocols including standardized packet formats and meanings of header fields cannot be changed; however, we can change behaviors of protocol stacks. For example, we can implement optimized packet transmission paths without firewall functions as a physical network protocol stack to relieve overhead due to tunneling. Even if a behavior of a physical network is changed, end-to-end data communications are preserved because the virtual network protocol stack supporting full features deals with them. Network policies are applied to communications on virtual networks. Physical networks must only deliver packets from source nodes to destination nodes.

Some might think that this architectural view is already known. Some implementations accidentally take the form of the separation; they have different network protocol stacks for virtual and physical networks. However, they do not exploit this potential for optimization. For example, network protocol stacks in VMs and HVs are isolated as shown in Figure 3.3. Network protocol stacks in VMs establish end-to-end connections, and network protocol stacks in HVs are responsible for physical packet transport. Nevertheless, network protocol stacks in HVs also support full networking features because the current design handles all packets by a single network protocol stack even if the packets are already encapsulated. The same is true of container technologies [49]. Containers are composed of kernel level isolation on user, process, file system and network spaces. Network spaces for each container and a host OS are isolated, but they are produced from identical implementation (codes) and support full features for packet handling. Therefore, existing implementations do not exploit the potential that the physical network protocol stacks after tunnels can be optimized.

Chapter 4

Improving Packet Transport at End Hosts

This chapter proposes a first implementation method that exploits the potential for optimization to improve performance with tunneling at end hosts. Tunneling that involves encapsulation causes processing overhead due to additional protocol processing compared to no encapsulation. In this chapter, we investigate magnitude of overhead and clarify bottlenecks. Then, we propose a new lookup method based on our architecture as an optimized physical network protocol stack for simple packet transport. This method achieves performance improvement at end hosts without modifications to existing protocols.

4.1 Introduction

A significant benefit from using virtual networking based on tunneling is the separation of physical networks and virtual networks. In network communities, virtual networks built on layer-3 tunneling are also called *overlay networks* [50]. Logical topologies of overlay networks isolated from physical IP networks can be changed agilely to respond to various demands. This characteristic is essential for today's virtualized environments such as VM based clouds and network function virtualization (NFV). However, overlaying networks requires additional protocol processing to transmit packets. This can lead to overheads due to the tunneling, especially in host operating systems.

A typical end point of overlay networks is software network stack of host OS. In clouds, hypervisors connect to overlays directly and virtual machines on the hypervisors are accommodated in the overlaid inner networks. In NFV, software-based middleboxes as service functions are connected by overlays [51]. Moreover, load balancers sometimes utilize overlays for isolation of virtual networks and physical networks [52, 36]. Tunneling protocols that build overlay networks are implemented as virtual network device drivers at the host OS. This virtual device, called virtual tunnel interface, becomes an entry point to an overlay network.

Network subsystems of host OSes are called network stack. Each layer protocol and interface is implemented carefully for code reusability and modularity to benefit from the layering model. Applications send data to network stack via socket API. Data is divided into packets by transport layer subsystems such as TCP or UDP implementations. Then, packets are processed by IP subsystem for routing, and delivered to Ethernet device drivers through Ethernet protocol implementation including ARP table lookup. The interface of the tunnel is the same as the link for data link or network layer protocols, so that tunneling protocols for Ethernet overlays are implemented as Ethernet device. Thus, IP subsystem does not need to mind that outgoing device is physical or virtual. When

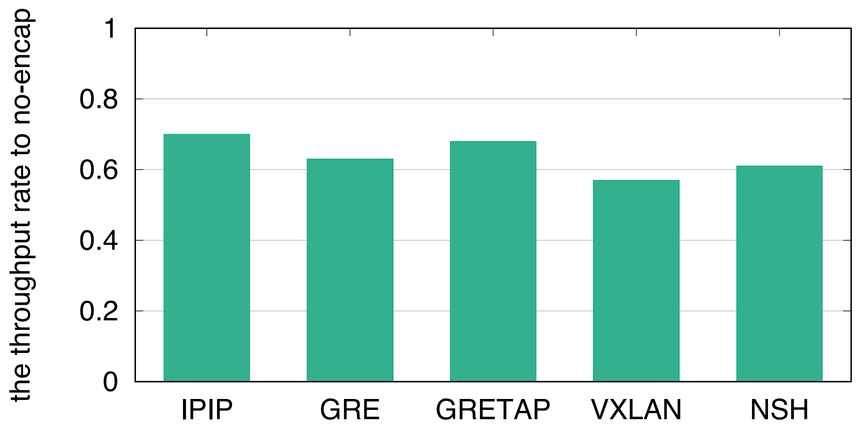


Fig. 4.1. The rate of packet transmission throughput via tunneling protocols compared with the normal transmission on Linux kernel 4.2.0.

the outgoing device is a tunnel interface, packets are encapsulated and put back to IP subsystem. Therefore, packets sent to a virtual tunnel interface **via a network stack** are encapsulated by the interface and transmitted to underlay physical networks **via a network stack again**.

This conventional OS network stack design of tunneling protocols involves overhead and degrades throughput because packets must pass through the same network stack twice. To demonstrate the degradation at the OS, we measured the packet transmission throughput with five tunneling protocols: IPIP, GRE, GRETAP (Ethernet over GRE), VXLAN and NSH[*1] on Linux kernel version 4.2.0 with Intel Core i7-3770K 3.50-GHz CPU and Intel X520 10-Gbps Network Interface Card (NIC). The test traffic was a single UDP flow generated by one CPU core in kernel space. Figure 4.1 shows the result of this experiment. The y-axis refers to the rate of transmission throughput via each tunneling protocol to the normal transmission (no encapsulation). As Figure 4.1 shows, using tunneling protocols in the host OS causes over 30% to 43% throughput degradation.

In this chapter, 1) we show a fine-grained bottleneck analysis of tunneling protocols on a Linux network stack. The resulting analysis shows the time required for each part of the network stack through tunneling protocols. Based on the analysis, 2) we propose a new lookup method for virtual networking with tunneling protocols. This method provides a shortcut for the second network stack processing. Furthermore, the method is protocol independent, so that the proposed architecture can be adapted to existing and future tunneling protocols. As a proof-of-concept, we implemented a part of the method using a commodity FPGA card and measured throughput improvement. Moreover, we implemented the proposed method into a software dummy interface and an Intel 10-Gbps NIC driver. The evaluation results show that the transmission throughput is improved in five tunneling protocols and the kernel throughput is approximately doubled in particular protocols.

[*1] Network Service Header has not implemented in Linux kernel network stack yet, and we used `https://github.com/upa/nshkmod`.

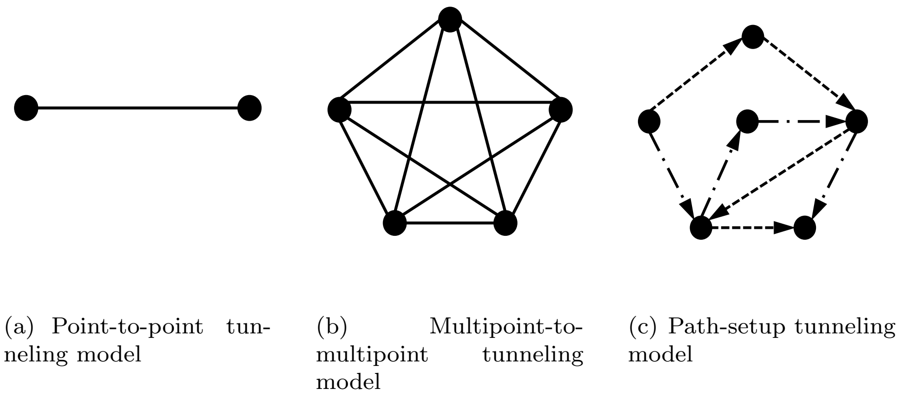

(a) Point-to-point tunneling model
(b) Multipoint-to-multipoint tunneling model
(c) Path-setup tunneling model

Fig. 4.2. Three topology model of tunneling protocols.

4.2 Case Study

Although new tunneling protocols are being proposed continuously for various purposes, all tunneling protocols can be classified into three models by logical topology type: point-to-point, multipoint-to-multipoint and path-setup. Point-to-point is a classic model that interconnects two hosts via a virtual wire as shown in Figure 4.2(a). Point-to-point tunneling is usually used to connect distant networks like Virtual Private Network (VPN). Typical protocols are IPIP and GRE. The multipoint-to-multipoint model constructs full-mesh topology between multiple hosts as shown in Figure 4.2(b). Multipoint-to-multipoint tunneling is used in cloud environments for example. All HVs connect to each other with the tunneling, so that VMs operated by a customer can be accommodated in a same virtual network even if VMs are placed among different HVs. A typical multipoint-to-multipoint protocol is VXLAN. Finally, Figure 4.2(c) shows path-setup model tunneling, which is used for NFV and service chaining. This model constructs virtual circuits on overlays. Network Service Header (NSH) is a typical tunneling protocol of the path-setup type.

We consider that the bottleneck of transmission (TX) path via tunneling protocols may change, based on the topology models. Multipoint-to-multipoint and path-setup model protocols requires some sort of table lookup to determine destinations on underlay physical networks corresponding to inner header of transmitting packets. For example, VXLAN has a forwarding table called VXLAN Forwarding Database (FDB). A VXLAN FDB entry is composed of a pair of destination MAC address of inner Ethernet frame and outer IP address of corresponding destination host. VXLAN finds the FDB to determine a physical network destination when encapsulating Ethernet frames. In contrast, point-to-point model tunneling has only one destination, so that it does not require such table lookup. Hence, we selected five tunneling protocols from three topology models as measurement targets: IPIP, GRE, GRETAP, VXLAN and NSH.

In order to clarify the bottleneck on the packet TX path via tunnels, we measured the time required for each part of the network stack processing to occur through five tunneling protocols on a current Linux kernel. Figure 4.3 shows the packet TX path of a Linux kernel and the experiment overview. The TX path through a tunnel is classified into three parts.

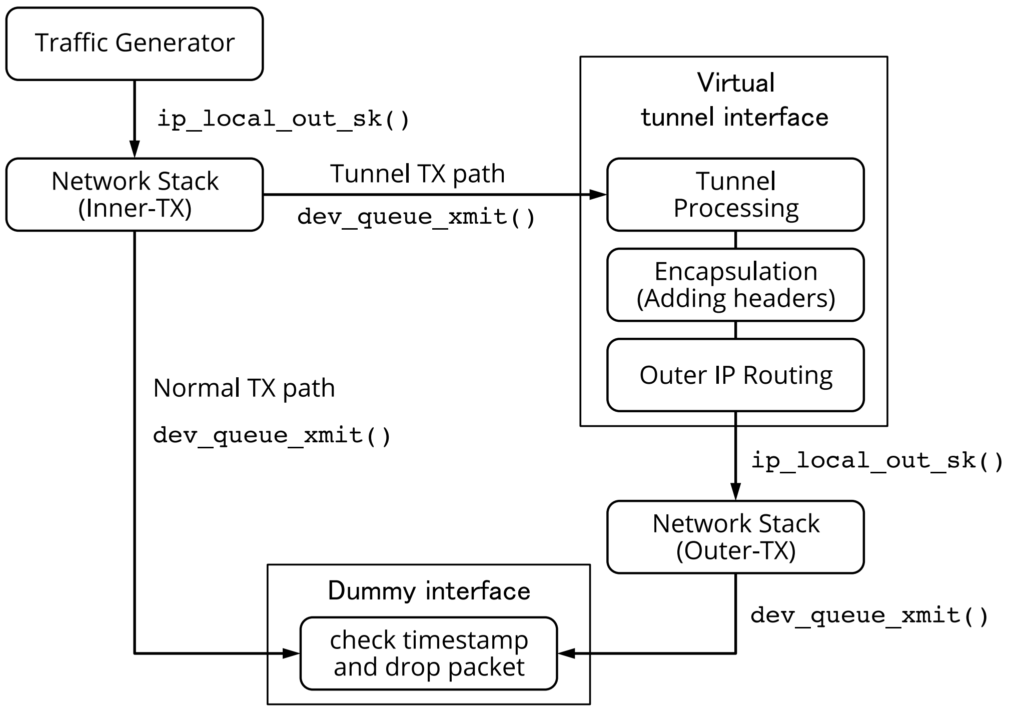


Fig. 4.3. Overview of TX path measurement on Linux kernel 4.2.0.

- **Inner-TX** starts from `ip_local_out_sk()` function. This processes transmitted packets after IP routing for layer-3 (e.g., fragmentation), layer-2 (e.g., ARP) and delivers them to a tunnel interface via `dev_queue_xmit()` function in the same manner as physical interfaces. The `dev_queue_xmit()` function is an interface of the physical layer in Linux kernel. Thus, output packets to physical or virtual devices uses this function in both cases.

- **Virtual Tunnel Interface** processes packets according to a tunneling protocol of this interface such as destination lookup on the tunnel and headroom allocation for outer headers (**Tunnel Processing**) and encapsulates the packets in the tunnel header and outer IP header (**Encapsulation**). After that, **IP Routing** runs for the outer destination IP address. Finally, the encapsulated packets are delivered to the network stack again via `ip_local_out_sk()`.

- **Outer-TX** processes the encapsulated packets for layer-3 and layer-2 and delivers them to an outgoing interface in the same manner as **Inner-TX**.

To measure the time required for these parts, we modified a Linux kernel. We added timestamp fields to the packet buffer (`struct sk_buff`), and inserted read time stamp counter (RDTSC) instructions to the start and end of each part. In this experiment, a packet generated by the traffic generator went through each part and transit times were stored in the timestamp fields of the packet buffer. The dummy interface shown in Figure 4.3 saved the timestamp fields and dropped the packet immediately when it received a packet. By using this method, we measured the time required for each part on the TX path with five tunneling protocols. For this experiment, we used the modified

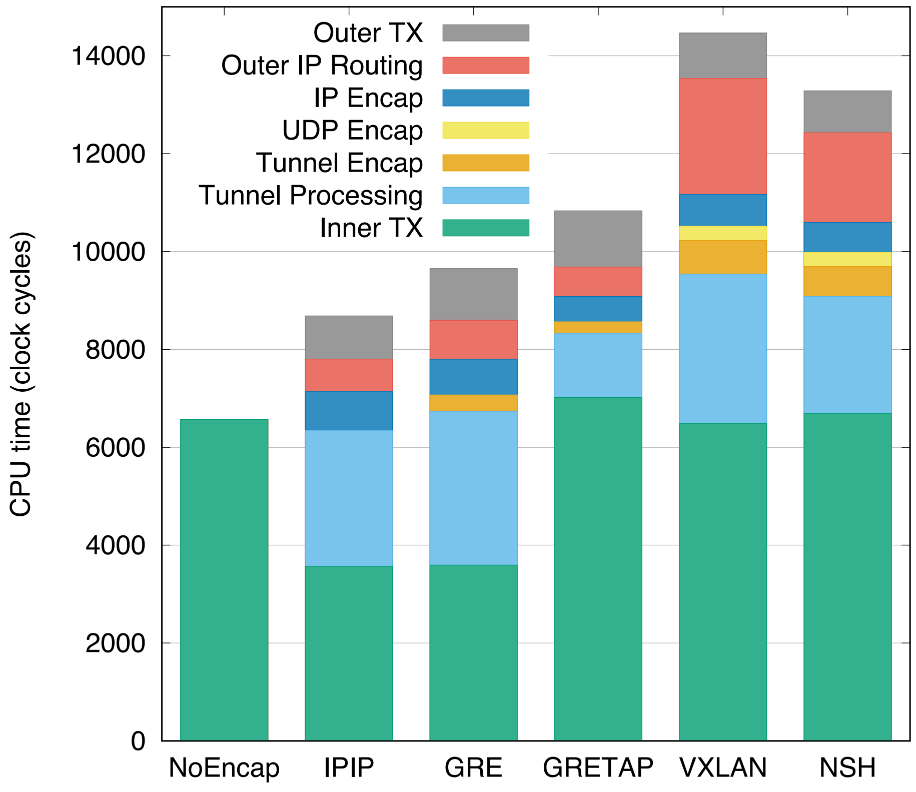


Fig. 4.4. The CPU time required for each part on the Linux TX path via five tunneling protocols.

Linux kernel 4.2.0*[2] with an Intel Core i7-3770K 3.50-GHz CPU machine and test packets were 64-byte UDP packets generated in kernel space. One core is used to generate test packets and the measurement. No applications using networks ran at the machine.

Figure 4.4 shows the result of this experiment. The y-axis refers to the time required as the number of CPU clock cycles. The value of each part is the median of the results of 300 runs for each protocol. The x-axis refers to tunneling protocols; NoEncap means the normal TX (no encapsulation), so that there is only Inner-TX on NoEncap. From the result, it can be seen that additional processing time due to tunneling cannot be ignored. In VXLAN and NSH, the CPU time required to transmit a packet is approximately doubled compared with the normal TX.

In VXLAN and NSH, the required CPU time for IP routing for the outer IP header (the red colored box in Figure 4.4) is larger than for the other cases. This is ascribable to differences of the topology models. IPIP, GRE and GRETAP are point-to-point models, so that the destination of the outer IP header is always the same. Therefore, the routing cache is used instead of outer IP routing. By contrast, in multipoint-to-multipoint (VXLAN) or path-setup (NSH) models, the destination IP address of the outer IP header is determined for each transmitting packet by their lookup mechanisms. IP routing cannot be omitted. Thus, the outer IP routing occupies over 16% of the TX path.

The reason why the Inner-TX of IPIP and GRE is shorter than other protocols is that packet copy due to headroom allocation for outer headers occurs in the Tunnel Processing instead of the Inner-TX. Layer-2 overlay protocols, GRETAP, VXLAN and

*[2] `https://github.com/upa/linux-madcap-msmt`

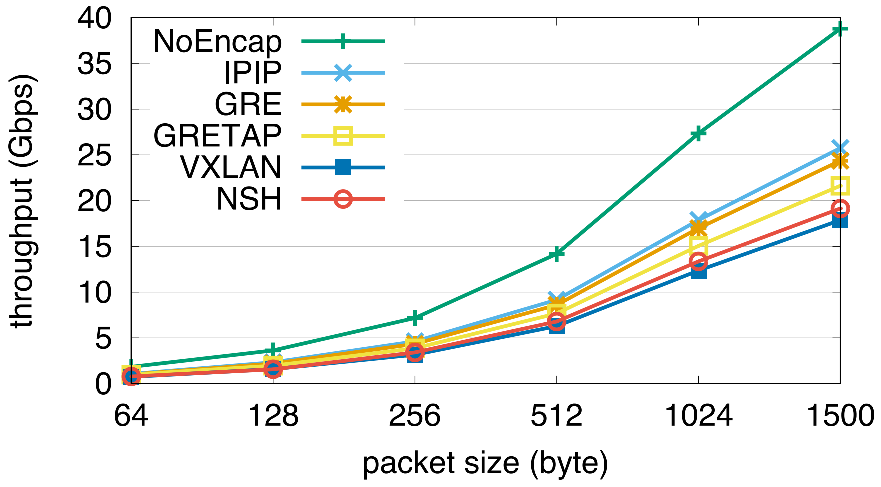


Fig. 4.5. Measuring TX throughput of Linux kernel 4.2.0 with a dummy interface.

NSH, allocate necessary headroom for the inner Ethernet header, tunnel header, outer IP and Ethernet headers together in the Inner-TX. On the other hand, layer-3 overlay protocols allocate headroom in the Tunnel Processing because the Inner-TX does not need to allocate headroom for the inner Ethernet header. As a result, the Inner-TX of IPIP and GRE is shorter than layer-2 overlay protocols, and Tunnel Processing of them is longer than others due to the position of headroom allocation.

GRETAP, VXLAN and NSH are layer-2 overlay protocols; however, required times for Tunnel Processing are different. This is caused by differences of topology models. GRETAP is point-to-point tunneling protocol, so that it can also use cache for outer IP routing. By contrast, VXLAN and NSH, multipoint-to-multipoint and path-setup model protocols must run lookup on the tunnel to determine destinations of inner packets in accordance with the table like VXLAN FDB.

In addition to the CPU time measurement, we measured TX throughput of the Linux kernel. In this experiment, the traffic generator kept generating test traffic and the dummy interface kept dropping the packets. Then, we counted the number of transmitted packets in 60 seconds. Test traffic of this experiment was a single UDP flow, and one CPU core was used to generate test traffic. Figure 4.5 shows the result of the experiment. Using tunneling protocols causes over 40% throughput degradation on the Linux kernel. Moreover, the result indicates that the current kernel network stack cannot make full use of the link speed. With 1500-byte packets, TX throughput of IPIP was 26 Gbps and VXLAN was 18.9 Gbps. These results are not sufficient for today's and future Ethernet device link speeds of 25 Gbps, 40 Gbps, 50 Gbps and 100 Gbps. As described, using tunneling protocols on a host OS involves additional overhead and highly degrades TX throughput.

4.3 Approach

In this study, we aim to shrink the gap of throughput with/without the tunneling protocols shown in Section 4.2. Before describing the approach, *protocol independence* has to be considered. New tunneling protocols are being proposed for various purposes on a daily basis. Thus, a method specialized in a particular protocol will soon be out of date. Moreover, such dedicated methods are difficult to deploy in the real world.

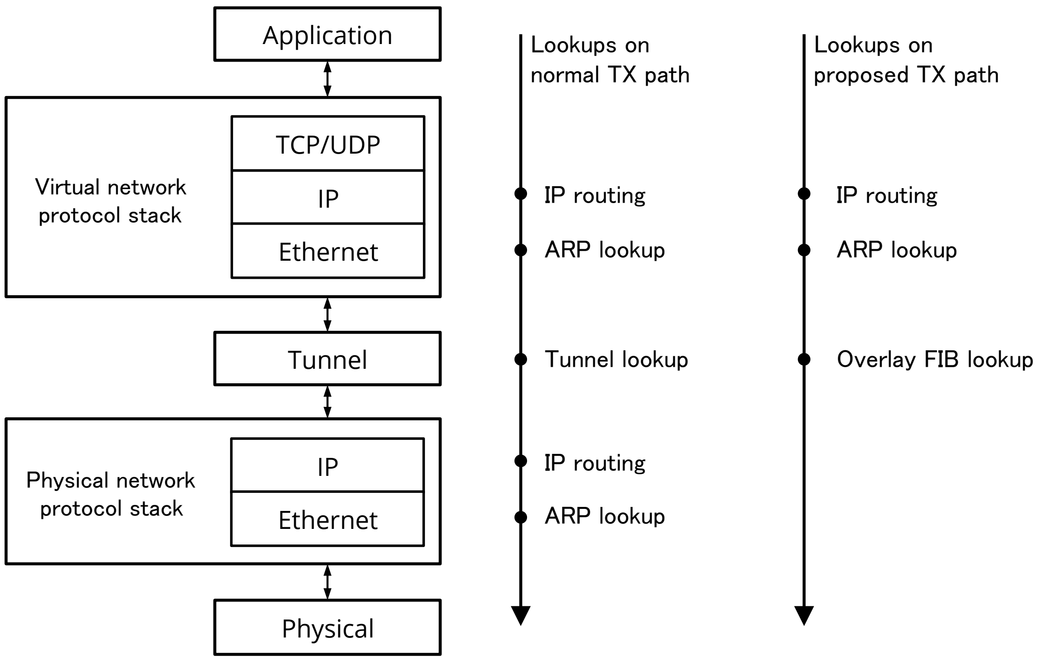


Fig. 4.6. Lookup events on the normal and proposed packet TX paths in the virtual network architecture: The overlay FIB omits second protocol processing including IP routing and ARP table lookups for the outer IP header processing.

Accordingly, we propose a protocol-independent forwarding information base (FIB) architecture for network overlays. Usual FIB is layer-3 FIB. The FIB is a forwarding table of IP routers to find proper destination MAC addresses and output interfaces for destination IP addresses of forwarding IP packets. To route and forward IP packet following layering model faithfully, IP lookup to determine a next-hop and an output interface and ARP table lookup to determine a destination MAC address of the next-hop are needed. FIB architecture transforms these two lookups into one lookup. The key of a FIB table entry is IP prefix, and the value of the entry is an appropriate destination MAC address and an output interface. By using the FIB table, packet forwarding is done by only one lookup for destination IP address.

We adapt this FIB architecture for a further upper layer that is overlay table lookup on the tunnel layer. As shown in Figure 4.6, there are three table lookups in the normal TX path via tunnels: 1) IP lookup for the inner destination IP address, 2) overlay lookup in virtual tunnel interfaces and 3) IP lookup for the outer destination IP address. The proposed method imports outer layer-3 and layer-2 header parameters from lookup tables into the overlay lookup. Entries of the proposed FIB, called overlay FIB, contains outer IP and MAC addresses. Therefore, outer IP lookup can be eliminated from the TX path because outer header parameters are found when the overlay FIB lookup runs.

The function of the overlay FIB is composed of two parts: 1) protocol-independent lookup for outer header parameters corresponding to a destination host and 2) encapsulating packets in outer headers. Figure 4.7 shows the TX paths with and without the overlay FIB. Tunnel interfaces deliver packets encapsulated in a tunnel header (e.g., GRE or VXLAN) to an outgoing interface directly, and then the function of the overlay FIB in the interface performs and encapsulates the packets in outer IP and Ethernet headers. Finally, the encapsulated packets are transmitted to a wire. The overlay FIB is placed in a part of NIC device driver software or NIC hardware. When the FIB is implemented in

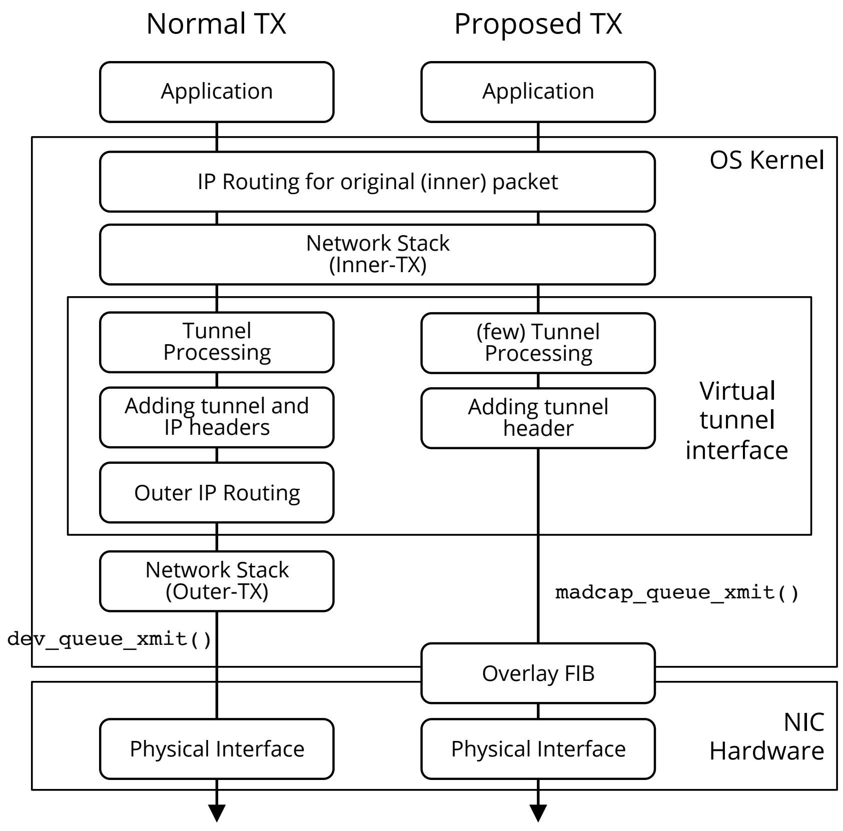


Fig. 4.7. The normal and the proposed TX paths through network overlays in the end host network protocol stack.

device driver software, eliminating second protocol processing including outer IP lookup saves the CPU time to transmit packets and improves throughput. When the FIB is implemented in NIC hardware, offloading the protocol-independent overlay lookup and outer header encapsulation to hardware improves throughput in addition to eliminating second protocol processing.

4.3.1 Protocol Independent Overlay FIB Lookup

Even if protocols or their topology models are different, the task of tunneling to transmit packets is the same: determining the destination and encapsulating the packets. Identifiers for determining the destination host that is the destination address of the outer IP header are different for each tunneling protocol. For instance, point-to-point protocols do not have identifiers because the tunnel has only one destination. Typical multipoint-to-multipoint protocols use the inner destination MAC address as an identifier to determine a destination. NSH, a path-setup model, determines a destination in accordance with the Service Path ID and Service Index embedded in the NSH header [44]. To handle these different lookup mechanisms as a single logic process for protocol independence, we define **offset** and **length** for the identifiers used in, for example, the BPF [53] approach.

Although identifiers for lookups are different for each tunneling protocol, they can be considered as the same type of processing: checking a particular byte-string embedded in a packet and determining a destination. Identifiers are always embedded in packets; therefore, identifiers can be specified by **offset** and **length**. The parameter **offset** indicates

Table 4.1. Identifiers embedded in packets can be specified by the offset and the length.

Protocol	Offset	Length	Identifier
IPIP, GRE	none	none	none (point-to-point)
VXLAN	16-byte	48-bit	Inner destination MAC address
NVGRE	8-byte	48-bit	Inner destination MAC address
MPLS over GRE	4-byte	20-bit	MPLS label
NSH over VXLAN-GPE	20-byte	32-bit	Service Path ID and Service Index

the beginning of the identifier and the **length** indicates the bit length of the identifier in the packets. Table 4.1 shows **offset** and **length** values for major tunneling protocols. In the case of VXLAN, **offset** is 16 bytes for UDP and VXLAN headers and **length** is 48 bits for destination MAC address. In the case of NSH over VXLAN-GPE, **offset** is 20 bytes for UDP, VXLAN and NSH base headers and **length** is 32 bits for Service Path ID and Service Index. In this manner, protocol-specific lookup mechanisms can be handled as a single operation on the overlay FIB.

An overlay FIB entry consists of a byte-string as an identifier and a destination IP address. In addition to the destination, necessary parameters for encapsulation are also stored: source IP address, IP header parameters including ToS and TTL, destination MAC address (gateway router's MAC address) and source MAC address. When transmitting a packet via a tunnel interface, the overlay FIB finds an entry corresponding to the identifier embedded in the packet. Thus, the packet is encapsulated in outer IP and Ethernet headers with parameters stored in the found entry from the FIB, and finally transmitted to a physical wire.

UDP encapsulation for several tunneling protocols can also be offloaded to the overlay FIB. Leveraging UDP encapsulation for tunneling [24, 26, 54, 55] is a typical technique to go through middleboxes that interfere and drop traffic except correct bidirectional TCP and UDP connections [56]. UDP encapsulation requires over 300 clock cycles as shown in Figure 4.4. The overlay FIB can accommodate UDP encapsulation functions. The UDP header is composed of source and destination port number, length and checksum. The destination port number is set in accordance with the tunneling protocol. The source port number should be calculated from a hash value of an inner packet for load balancing on intermediate node such as Equal Cost Multipath. The length and checksum fields must also be calculated for transmitting packets. If the overlay FIB has functionality for calculating them and storing the destination port number in the entries, UDP encapsulated can be offloaded. It is feasible because that these functions can be implemented as software easily, and hardware that are capable of such offload exists as the UDP tunnel offload [39, 38].

The protocol-independence of our method contributes to code reusability and a likelihood that the FIB is implemented in NIC hardware. New networking protocols are being proposed on a daily basis; therefore, hardware ossified to particular protocols will soon necessitate updating or be obsolete. The software case is also the same. Networking features are implemented in OS kernel space in general; however, developing kernel code requires expertise and it leads to barrier [57, 58, 37]. The overlay FIB lookup method is protocol-independent because the FIB does not consider protocol context of the identifiers that means the identifiers are handled as just byte-strings. Therefore, it can be adapted to past, current and feature tunneling protocols.

4.3.2 Design and Implementation

Basic overlay FIB update operations are adding or deleting a FIB entry composed of an identifier and a destination IP address. Some sort of control plane systems can trigger the operations. For example, EVPN based on Border Gateway Protocol (BGP) can perform a control plane for the VXLAN overlay [59]. A control plane system can learn a pair of inner MAC address and destination node IP address via BGP and add the pair as an overlay FIB entry. When an overlay FIB entry is added, IP routing table lookup and ARP table lookup for the destination IP address run immediately. Then the entry forms a pair of an identifier that is the inner MAC address and outer header parameters.

The overlay FIB is also updated by state changes of under layers shown in Figure 4.6. This is similar to IP routing table change due to the link up or down. When a link status becomes down, corresponding ARP entries and IP route entries are deleted. Then, the overlay FIB must detect the changes and update corresponding FIB entries by looking up IP routing table and ARP table in the same manner as the update operations. The FIB must also detect IP routing table and ARP table changes to maintain entries.

4.3.2.1 Hardware implementation

We implemented a part of proposed functions in a prototype NIC using a NetFPGA-1G card [60] with Linux. Our NIC implementation is still not matured yet; however, we confirm that it is possible to develop our technique on commodity NIC hardware. This implementation has the only function for the offloading outer IP encapsulation without the table lookup. The prototype consists of the NetFPGA reference NIC design and our encapsulation module. The module adds an outer IP header with specified source and destination IP addresses, TTL, ToS and protocol number.

In order to support the prototype NIC, we modified IPIP and VXLAN drivers on Linux. In Linux kernel, the tunneling protocol drivers call the `ip_tunnel_xmit()` function that processes outer IP encapsulation, IP routing and transmitting. This function in the drivers is replaced with the `nf2c_tx()` function that enqueues a packet to the TX buffer of the NetFPGA NIC. Our IPIP driver places IP packets to the NIC TX buffer directly, and our VXLAN driver places an Ethernet frame encapsulated in UDP and VXLAN headers to the buffer through the `nf2c_tx()`.

4.3.2.2 Software implementation

We implemented the whole of overlay FIB as software modification for ixgbe Intel 10-Gbps NIC device driver on Linux kernel version 4.2.0. The modified ixgbe driver provides a new packet entry point named `madcap_queue_xmit()` for tunnel interfaces instead of `dev_queue_xmit()` as depicted in Figure 4.7. When the driver receives a packet encapsulated in a tunnel header from the entry point, the overlay FIB lookup and encapsulation perform for the packet. Then, the encapsulated packet is delivered to NIC hardware. Moreover, the driver detects IP routing table change through switchdev API [61] and ARP table change through the netevent notifier.

The driver also provides a configuration interface for the overlay FIB. The configuration interface for the overlay FIB, called madcap API, provides add, delete and show operations. The API is driver independent. Device drivers supporting madcap has `struct madcap_ops` structure in their `struct net_device` that represents the network interface in Linux kernel. `struct madcap_ops` has functions for offset and length configuration, UDP configuration, entry insertion and deletion, dump configurations and dump entries. Userland application such as control plane systems can call the functions via Netlink [62] independently to protocols and devices. We implement a modification for iproute2 [63], a popular network configuration command package on Linux, to control the FIB via `ip` commands like the following examples.

```
Command 1: configuring a madcap capable device
  $ ip madcap set dev eth0 offset 16 length 48 src 172.16.0.1 proto udp

Command 2: show configuration of a device
  $ ip madcap show config
  dev eth0 offset 16 length 48 proto 17 src 172.16.0.1

Command 3: entry insertion and deletion
  # add entry, key is MAC address 3c:15:c2:c4:f5:88 and dst is 10.0.0.1
  $ ip madcap add id 3c15c2c4f588 dst 10.0.0.1 dev eth0

  # del entry, key is MAC address fe:54:00:db:0e:40 and dst is 10.0.0.2
  $ ip madcap del id fe5400db0e40 dst 10.0.0.2 dev eth0
```

We modified five tunneling protocol drivers, IPIP, GRE, GRETAP, VXLAN and NSH drivers to support the overlay FIB. The modification is inserting the `madcap_queue_xmit()` function to TX path for the case that underlay device is a madcap capable device. All modifications for each protocol driver are less than a few dozen lines of code. Source codes described in this chapter are available [*3].

4.4 Evaluation

The proposed overlay FIB architecture provides a shortcut for second protocol processing on the TX path via tunneling protocols. By using the proposed architecture, TX throughput with tunneling protocols is improved. In this section, we investigate throughput improvement due to the proposed architecture from three aspects: 1) effect of offloading IP encapsulation in NIC hardware, 2) the required CPU time to transmit a packet and 3) actual throughput with software implementation.

4.4.1 Effect of offloading IP Encapsulation on Hardware

First, we measured transmission throughput using the prototype FPGA NIC implementation. The NIC has the function that encapsulating packet in specified outer IP and Ethernet headers. This result suggests an effect of the offloading per-packet IP encapsulation from software to hardware. This means eliminating the blue and red boxes (IP Encap and Outer IP Routing) on Figure 4.4. We measured and compared 64-byte packet transmission performances with IPIP and VXLAN protocols in packet per second as shown in Table 4.2. The experiment platform was Linux kernel 3.16 and Intel Core i7-3770K CPU.

When IPIP encapsulation is offloaded, the throughput reaches the no encapsulation throughput. Our IPIP driver carries packets to the NIC via the `nf2c_tx()` immediately after a few error checks. There is no overhead for the `ip_tunnel_xmit()` that performs IP encapsulation and outer IP routing. In the case of VXLAN, the offloading also contributes to the throughput improvement; however, the throughput is lower than the no encapsulation because the destination lookup for VXLAN before the `nf2c_tx()` is not offloaded in the prototype implementation. As a result, the IP encapsulation is overhead, and the offloading it on NIC is a credible way for the throughput requirement.

In addition, IPIP with the offload is faster than NoEncap. The reason for this incomprehensible is that IPIP is a layer-3 overlay protocol. As described in Section 4.2, layer-3 overlay protocols do not allocate headroom for outer headers including Ethernet header.

[*3] `https://github.com/upa/madcap`

Table 4.2. Measuring transmitting throughput with IP encapsulation offload (kpps).

packet size (byte)	NoEncap	IPIP (offload)		VXLAN (offload)	
		off	on	off	on
64	112.00	106.88	114.51	97.52	106.53
1024	29.64	27.68	30.00	27.55	27.33

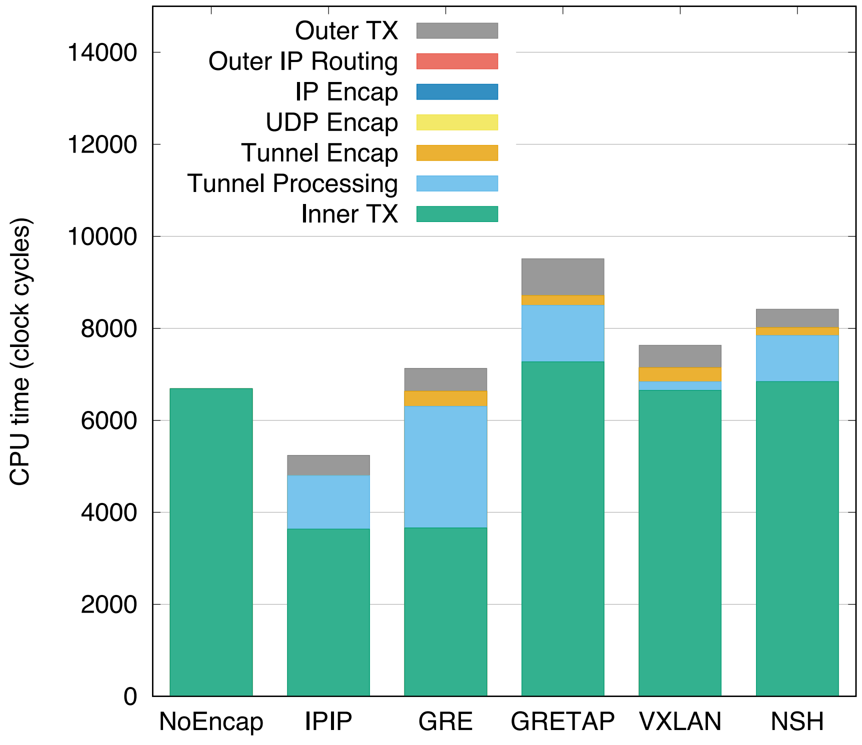


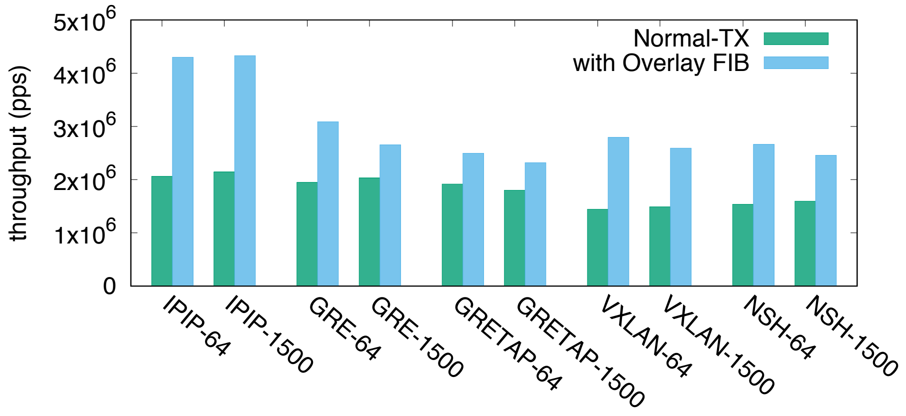
Fig. 4.8. Outer UDP, IP encapsulation and IP routing are omitted from the TX path compared with the normal TX path shown in Figure 4.4.

Thus, there is no packet copy in the case of IPIP with the offload in contrast to VXLAN that is a layer-2 overlay protocol.

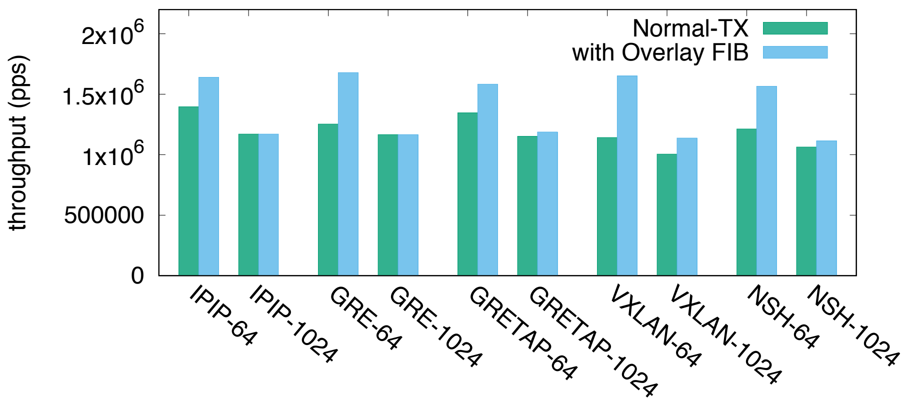
4.4.2 Measuring CPU time on the TX path

Next, we measured the CPU time required for each part on the TX path with the overlay FIB in the same manner as for the experiment discussed in Section 4.2. In this experiment, we used the dummy driver. Packets with a tunnel header delivered from a modified tunnel interface were dropped immediately in the dummy interface. That is, results of this experiment show the pure kernel throughput if the overlay FIB was completely implemented in NIC hardware. We experimented with the modified Linux kernel version 4.2.0 and an Intel Core i7-3770K 3.50-GHz CPU machine.

Figure 4.8 shows the CPU time required to send a packet in kernel if the FIB was implemented in NIC hardware. In Figure 4.8, Outer-TX means the time required from the end of tunnel interface to the start of dummy interface processing. IP routing, IP and UDP encapsulation are completely removed from TX paths compared with Figure 4.4. As a result, the CPU time required to send a packet in the kernel is reduced in all tunneling protocols. In addition, IPIP with the FIB is shorter than NoEncap. That is also because



(a) TX throughput with the dummy interface.



(b) TX throughput with modified ixgbe driver.

Fig. 4.9. The measurement results of the experiment on TX throughput.

of the packet copy for headroom allocation. In the normal TX path of IPIP, headroom for outer IP and Ethernet headers is allocated in the Tunnel Processing as described in Section 4.2. By contrast, allocating headroom is not needed if outer encapsulation is offloaded to NIC hardware by the proposed method. As a result, CPU time for IPIP becomes shorter than for no encapsulation, which requires headroom allocation for the Ethernet header in the Inner-TX. On the other hand, time for the Tunnel Processing of GRE, which uses layer-3 tunneling, does not decrease, because headroom allocation is still needed for the GRE header.

In VXLAN, the Tunnel Processing is almost removed. When using the FIB, protocol processing of VXLAN only involves adding a VXLAN header to a packet. Lookup VXLAN FDB, determining destination IP address, encapsulating VXLAN and UDP headers are removed from the Tunnel Processing. As a result, the CPU time required to transmit a packet via the VXLAN tunnel with the FIB has decreased by 47% compared with the normal VXLAN.

4.4.3 Measuring Throughput

Finally, we evaluated throughput using the dummy interface and the modified ixgbe driver, which is the Intel 10-Gbps NIC driver. With the dummy interface, transmitted packets were dropped immediately the packets were delivered to the dummy interface; this is the same as for the experiment shown in Figure 4.5. Altogether, the throughput of the dummy interface means the throughput of the Linux kernel itself. In addition, we implemented the overlay FIB as software into an ixgbe driver. With the modified ixgbe driver, packets delivered to an ixgbe interface were encapsulated in outer IP and Ethernet headers in accordance with the overlay FIB in the device driver. Then, the packets were transmitted to a wire via X520 NIC. We measured throughput of the modified ixgbe at a receiver machine connected to the NIC. For this experiment, we used the same machine as that used in the experiment described in Section 4.4.2.

Figure 4.9 shows transmission throughput using the dummy interface and the modified ixgbe driver. The y-axis indicates transmission throughput in packet per second. The x-axis indicates tunneling protocols and packet size used in this experiment. With the dummy interface, we measured throughput with 64-byte (green boxes) and 1500-byte (blue boxes) packets. With the modified ixgbe driver, we measured with 64-byte (green boxes) and 1024-byte (blue boxes) packets because transmission throughput with 1500-bytes was over 10Gbps that is the link speed of X520 NIC.

In all cases, the proposed method improves TX throughput. Throughputs of the dummy interface with IPIP and VXLAN with 64-byte packets are also approximately doubled as, expected from the result of the experiment about CPU time in Section 4.4.2 On the other hand, the overlay FIB software implementation also improves actual throughput to the wire when using the ixgbe driver in all tunneling protocols. When it was implemented in the NIC hardware, the throughput of the ixgbe was improved at the same rate as for the dummy interface.

4.5 Discussion

■Other identifiers: The proposed method assumes that the identifier is embedded in the packet and is just a byte-string. This assumption is correct for MAC addresses or labels; however, it does not work for identifiers that have some sort of semantics. Locator Identifier Separation Protocol (LISP) [43] utilizes the inner destination IP address as the identifier to determine a destination on a LISP overlay network. Thus, the overlay FIB has to be capable of the longest prefix match in order to support LISP.

■Other bottlenecks on the TX path: Many bottlenecks on the TX path are well known. For instance, socket API, system call overhead and per-packet processing have been mentioned for high-speed packet I/O approaches [64, 65, 66, 67]. Meanwhile, network protocol processing never disappears from host OS. This means, no matter how fast the network stack is, protocol processing becomes the bottleneck on the TX path. In the Arrakis operating system [68], which has a very optimized network stack, network protocol processing occupies 44.7% of its TX path. When protocol processing becomes a bottleneck as it does in Arrakis, the current design of overlays where packets go through the network stack twice involves serious overheads. Therefore, the proposed method, which omits the second protocol processing, can achieve throughput improvement even if other bottlenecks are also relieved.

■Lack of packet handling features: Our proposed method eliminates second protocol processing. Hence, existing packet handling features in OS network stack cannot be applied to encapsulated packets. For instance, Linux has many features for packet handling such as firewall and NAT; however, they do not act on encapsulated packets. This is a shortcoming of the proposed method. By contrast, these features are typically needed by virtual machines or middleboxes accommodated in overlay networks. Our method does not change first protocol processing; thus, these packet handling features act on inner packets. To handle packets after the overlay FIB encapsulating them in device drivers or NIC hardware, NICs have to support such features in hardware [69, 70].

■Hardware Offloading Techniques: TCP segmentation offload (TSO) and generic segmentation offload (GSO) are well-known techniques to improve throughput on host operating systems. They improve TX throughput by reducing the number of packets that are handled in the kernel network stack by dividing large packets into MTU sized packets. By contrast, the proposed method improves throughput by reducing processing for each transmitting packet. There is no collision between them, so that they can coexist. Some NICs [39, 38] has TSO feature for VXLAN encapsulated packets, called UDP tunnel segmentation. Inserting the overlay FIB function as both software and hardware before TSO, large encapsulated packets can be divided into small packets the same as the normal use of the offloading. SoftNIC [71], a flexible software NIC design, also achieves high performance via VXLAN tunneling using TSO. In this case, by implementing and inserting the FIB as a module before the TSO module (between vport_inc and parser in the SoftNIC pipeline example), the proposed method can be adapted to SoftNIC.

■RX path: This study is less concerned with RX paths. In TX path, outer IP and MAC addresses can be determined proactively by IP routing table and ARP table lookups if a destination IP address corresponding to an identifier is given by control plane systems. However, in RX paths, receivers never know what kind of packets will come. Receivers must parse received packets from the beginning. Therefore, the proposed method cannot be adapted to RX paths.

4.6 Summary

Tunneling for virtual networking will degrade TX throughput at the end host due to additional protocol processing. We have investigated the bottleneck of using tunneling protocols in the Linux kernel network stack. Based on the investigation, we have proposed a new FIB architecture for virtual networking. The overlay FIB provides a shortcut for second protocol processing of tunneling protocols. Moreover, this architecture is protocol independent, so that it can be adapted to existing and future tunneling protocols. This independence provides code reusability and potential for offloading the FIB on hardware. We demonstrated that offloading outer IP routing and encapsulation was effective for throughput improvement by using the prototype FPGA implementation, our proposed method could improve transmission throughput with five protocols from two viewpoints: CPU time required to transmit a packet and actual throughput. The proposed method decreased the CPU time required to transmit a packet by over 12% in general, and particularly decreased the CPU time by 47% in VXLAN tunneling. Furthermore, the kernel throughput was approximately doubled in IPIP and VXLAN.

The main contribution of this research is that the FIB approach across layer boundaries improves performance on stacked layers. Our FIB architecture that merges lower layer lookup tables into the tunnel layer lookup is not limited to existing tunneling protocols. In addition, lower network and data link layer protocols are also not limited to IP and ARP mentioned in this dissertation. When a new network protocol emerges, its routing table will be able to be merged into the FIB. Moreover, this approach can be enhanced to layers above the virtual network. For example, a socket of the transport layer specifies destinations with 5-tuple (protocol number, source, destination IP addresses and port numbers). Thus, if sockets have such FIB, lower layer lookups including network, data link, and tunnel layers will be omitted. Overall, our overlay FIB approach can improve performance on layering models.

From the viewpoint of the virtual network architecture, this chapter presented a method to exploit the potential for optimization. This method is based on the view that physical networks for the locator aspect are separated from data communication context, so that the physical network protocol stack can shrink for optimization. In the overlay FIB architecture, the identifier aspect is preserved in the virtual network protocol stack. Inner IP addresses processed through the Inner IP routing and the Inner-TX perform host identifiers for end-to-end data communications. Meanwhile, the overlay FIB performs the locator aspect as simple packet transport. The FIB determines locators in accordance with identifiers of inner packets such as MAC addresses, and transmits the encapsulated packets to physical networks. The overlay FIB achieves this optimization by enhancing functions of tunneling: protocol-independent lookup method on the tunnel shim layer as determining destination, and adding lower layer headers simultaneously as encapsulation. The significance of this research is that we have solved performance degradation at end hosts due to tunneling based on the separation of identifier and locator aspects.

Chapter 5

Improving Packet Transport at Networks

This chapter proposes a second implementation method that exploits the potential for optimization to improve performance at networks. We demonstrate a method that realizes explicit path control based on our architecture by an optimized physical network protocol stack via the novel usage of tunneling. This method changes a behavior of physical network protocols: end hosts add multiple locators to a packet to identify a path in a physical network. Packets are delivered across multiple paths using multiple links efficiently. Network devices do decapsulation in addition to usual IP routing and forwarding. Accordingly, this method achieves performance improvement at networks without modification to commodity network devices.

5.1 Introduction

Improving the availability and throughput of networks is quite an important topic for data center environments in particular. For a data center network, availability means being able to adapt flexibly to topology changes due to failures. During the operation of a service network, the topology may change from time to time for a variety of reasons, including maintenance, enhancements and breakdowns. The data center network should maintain its services while adapting flexibly to these topology changes. In addition to maintaining its availability, improvements to the throughput of a given network must be achieved. With the emergence of cloud computing with virtual networking and distributed applications, traffic in data centers has been growing rapidly. Nevertheless, administrators cannot simply deploy additional hardware to increase capacity on every occasion because of cost. Therefore, improving the aggregated bandwidth of a given network is an important challenge for data centers.

To address these challenges, many approaches have been proposed by both operator and research communities. A hop-by-hop routing architecture with dynamic routing protocols is a good way to deal with the availability issue. When the topology is changed, layer-3 routers detect the change and recalculate the routing tables, following which the network recovers autonomously. The hop-by-hop routing architecture is not necessary to consider the totality of the network. Effectively, this approach localizes the impacts of topology changes. The use of the traditional IP network is a proven technique from the viewpoint of availability. Layer-3 routing protocols such as OSPF [72] and BGP construct hop-by-hop routing tables autonomously at all routers. Constructing data centers using layer-3 networks to support availability has been proposed and developed [34, 35]. In addition, some methods introducing this architecture to layer-2 networks have been proposed [29, 30] for the availability issue.

To improve the throughput of a given network, there are many approaches applicable to data center networks. One popular technique is the use of multipathing [73]. Traditional IP routing and forwarding follow sink tree and the shortest path forwarding principle as mentioned earlier. Multipathing is that packets from a source to a destination are taken through multiple paths. Consequently, aggregate throughput of a network is improved by using extra links excluded from shortest paths. Prior work on topologies that have alternative paths between arbitrary hosts includes fat-tree [32], random graph [33] and more topologies [74, 75, 76, 77]. Furthermore, various approaches improving throughput by multipathing via these topologies have been proposed [73]. A classic and practical way to achieve multipathing is Equal Cost Multipath (ECMP). ECMP-enabled routers divide traffic into multiple next-hops by considering hash value of flows. Alternative schemes achieve efficient multipathing by introducing special addressing schemes and modified hardware to both layer-2 and layer-3 networks [32, 31].

Although existing approaches achieve good availability and/or throughput via multipath topologies, many of these approaches have not yet to be deployed. Economic cost is a significant problem for data centers [78], therefore, administrators strongly want to use commodity products on most occasions. However, existing proposals for availability [29, 30] and throughput improvement by multipathing [32, 31] require some new hardware that is not commercially available. This is a barrier to deployment and increases the cost of building a network. To achieve good throughput while using commodity hardware, ECMP is the usual approach [34, 35, 36]. However, it is known that ECMP cannot use multiple paths efficiently [32]. Accordingly, there is no current method that can achieve optimal both availability and optimal throughput while using commodity routers.

In this study, we focus on multipathing for a layer-3 data center network with unmodified commodity, commercial off-the-shelf (COTS) devices. Layer-3 multipathing offers both availability and throughput improvement, and using COTS devices eliminates barriers to deployment. To achieve this aim, we propose an explicit path control method by using tunneling protocols. Current COTS devices like Top-of-Rack (ToR) switches are capable of layer-3 routing and some tunneling protocol decapsulation on their hardware. In the proposed method, end hosts encapsulate packets in one or more tunnel headers, and the packets are taken through explicit paths represented by specified relay routers in the data center network. End hosts manage path status and traffic balancing so that any modifications for network devices are not needed. Therefore, we can shift the functionality of traffic balancing from the network switches to the end hosts in the data center. We designed and implemented the required modification of end-host software. In other words, instead of using a source route option, we use an IP encapsulation technique as an identifier to specify paths of multipath.

In order to demonstrate multipathing by our method, we applied the method to two topologies: a fat-tree topology and a random graph. In the fat-tree topology, we adapt a flow assignment algorithm proposed by Al-Fairs [32] for traffic load balancing for our method. In random graph topology, we integrated our method with k-shortest path algorithm and MultiPath TCP [79] proposed by the previous work [33]. In addition, we design and implemented a control plane system using Open Shortest Path First (OSPF) that is a popular layer-3 routing protocol for high availability. We evaluated the proposed method for both fat-tree and random graph topologies. The results show that our approach achieves a forwarding rate of 100% for a particular traffic pattern and over 85% for all traffic patterns in the fat-tree topology. The other results show that the proposed approach achieves higher throughput than ECMP on the random graph topologies in our simulation. These are comparable to the performance of existing proposals that require dedicated hardware.

5.1.1 Terminology

We adopt the following terminology for intermediate nodes in this chapter. An intermediate node from a part of a data center network is a *switch* even if that performs the same role of layer-3 routers. In contrast to conventional layer-2 switches, current ToR switches, called *layer-3 switches*, are comparable to routers from the viewpoint of layer-3 routing and forwarding functionalities.

5.2 Related Work

Various approaches addressing issues for data center networks have been considered, proposed, and developed. Table 5.1 shows a comparison of characteristics of prior work based on SPAIN [4] work. We have added a Topology Change column to the table, which indicates availability under topology changes. Based on Table 5.1, we discuss and summarize previous work from the viewpoint of availability, throughput improvement and deployability due to using COTS switches.

The most important feature of topologies for data center networks is that there are multiple paths between arbitrary two hosts. Balancing traffic among the multiple paths achieves throughput improvement. One of the most popular way for this multipathing is ECMP. ECMP enabled devices balance forwarding traffic among multiple equal cost next-hops via multiple links. The key to determine a next-hop for a packet is hash value calculated from a combination of source, destination, MAC, IP addresses and port numbers of the packet. Typical COTS switch products support ECMP and are deployed [34, 35, 36]; however, ECMP has two problems. First is that ECMP does not account for flow bandwidth, which can lead to oversubscription on particular links [32]. Second is that ECMP can only use *equal cost* shortest paths, hence, it cannot use sufficient usable paths and links in particular topologies such as random graph [33].

As mentioned above, a layer-3 network localizes the impact of topology changes by using hop-by-hop routing with dynamic routing protocols. Facebook [35] and VL2 [36] achieve their availability by constructing a backbone network based on a layer-3 network. In addition, both approaches use COTS switches, which support deployability. However, they use inefficient ECMP to improve throughput on a multipath topology.

In contrast to ECMP, SPAIN [4] uses VLAN for multipathing on an arbitrary topology with COTS switches. SPAIN proposes a greedy algorithm to identify each path in a VLAN. The path state remains on the network, and end hosts split traffic into multiple paths for identified VLANs. SPAIN achieves both good deployability and throughput improvement. However, recomputing and reconfiguration are required when the topology is changed such as switches being added or dropped. In addition, XPath [80], a technique for multipathing in commodity-based layer-3 networks, also leaves the path state on the network using longest prefix match table instead of VLAN of SPAIN. Accordingly, they lack the characteristic of availability.

TRILL [29] and SEATTLE [30] offer availability by introducing a hop-by-hop routing architecture to a layer-2 network. However, they require dedicated hardware for the network switches, resulting in a deployability barrier. On the other hand, PortLand [31] achieves throughput improvement with multipathing by introducing a dedicated address assignment scheme and topology restrictions. The dedicated addressing scheme for multipathing on an IP network was first proposed by Al-Fares *et al.* [32]. PortLand updates this scheme for the MAC address of Ethernet. However, it also requires the installation of dedicated hardware switches.

Table 5.1. Comparison of related work based on SPAIN [4].

	Topology	Usable paths	COTS switches	Topology change
Facebook [35]	Multiple Tree	ECMP	YES	YES
TRILL [29]	Arbitrary	ECMP	NO	YES
SEATTLE [30]	Arbitrary	Single Path	NO	YES
PortLand [31]	Fat-tree	ECMP	NO	NO
SPAIN [4]	Arbitrary	Multiple Paths	YES	NO
VL2 [36]	Fat-tree	ECMP	YES	YES
Jellyfish [33]	Random Graph	Multiple Paths	NO	-
Our method	Arbitrary	Multiple Paths	YES	YES

Finally, Jellyfish [33] proposed by Singla *et al.* uses random graph as a data center network topology. Random graph is better than fat-tree in specific cases such as building cost and throughput in failures. The number of equal cost shortest paths between two end hosts in random graph is less than other topologies. Thus, Jellyfish utilizes k-shortest paths and MultiPath TCP [79] at end hosts to split traffic on k-shortest paths. However, k-shortest path routing is not common in hardware. As a result, Jellyfish is not feasible for commodity-based network due to need for some new hardware features for k-shortest path routing.

Then, we propose a method for achieving multipathing in a layer-3 network that is different from ECMP with unmodified COTS switches. It provides availability as a benefit of using the layer-3 network, throughput improvement by using a multipathing method and deployability by using COTS switches.

5.3 Approach

We propose the method on the assumption that the commodity data center network is based on IP routing and forwarding technologies that are already available via COTS switches. Enabling a layer-3 network design provides both availability and deployability. In the proposed method, the end hosts select a path among multiple paths for each flow of traffic. Hardware COTS switches do not have a functionality of efficient traffic balancing, and ECMP does not consider the bandwidth of an individual flow which can lead to oversubscription. We therefore shift the functionality of traffic balancing from the network switches to the end hosts, similarly to SPAIN. However, in contrast to SPAIN, the end hosts use the IP encapsulation technique supported by COTS switches to specify path in the layer-3 network.

Figure 5.1 shows the overview of the proposed method. Each path from a host to others is specified by a corresponding relay switch. In Figure 5.1, when the shortest path from switch A to D passes through B, the end hosts can identify an extra path by specifying switch C as the relay switch, and split their traffic by specifying either switch B or C for packets. Therefore, if an identifier is added to a packet that specifies the address of the relay switch, an end host can select a path explicitly. This idea is essentially the same as the source routing.

To route a packet through a specified relay switch, the source route option can be used. Source route options, IPv4 loose source and record route option (IP option type 131) and IPv6 routing header (IPv6 Next header type 43), are standardized as a part of the IP stack specification, and almost all COTS switches therefore support them routinely. When a host sends a packet to another host, the sender can add a source route option with the

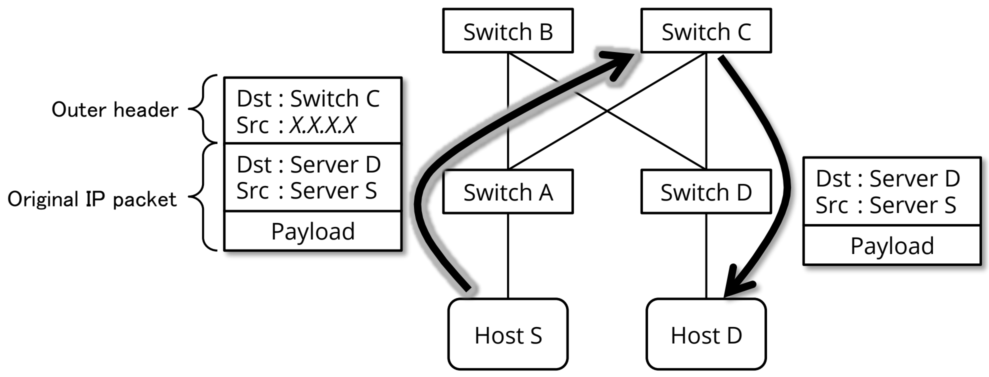


Fig. 5.1. How to specify the extra path using encapsulation techniques.

address of the corresponding relay switch to specify a through path explicitly. However, the source route option causes degradation of the forwarding performance because it is usually processed by the CPU. The source route option is not suitable in practice.

Instead of using a source route option, we use an IP encapsulation technique as an identifier to specify the relay switches. In the proposed method, the end hosts add a tunnel header and encapsulate it in an outer IP header when sending a packet to other hosts. The destination address of the inner header is the destination host, and the source address of the inner header is its own address. Further, the destination address of the outer header is the relay switch, and the source address of the outer header is a proper address that does not exist in the network. As shown in Fig 5.1, host S sends a packet to host D encapsulated in an outer IP header with switch C as the destination. When the packet is received by switch C, the outer IP header and tunnel header are removed, with the original packet then being forwarded to the original destination, host D. In this way, end hosts can specify a path from a set of multiple paths by using tunneling protocols. Note that the encapsulation technique will incur throughput degradation because of the reduction of packet size. However, this may not be significant because the jumbo frame is often adopted in data center environments.

When using tunneling protocols for this approach, a tunnel scalability problem may occur. To utilize all links in a data center network efficiently, each end host should be able to specify any relay switch. The number of tunnels will then be given by the number of $hosts \times relayswitches$, whereas COTS switch hardware supports at most hundreds of tunnels. Therefore, this will not be suitable for large-scale data center network that accommodates thousands of servers. The proposed method avoids this scalability issue by using a *non-existent address* for the source address in the outer IP header. The $X.X.X.X$ shown in Figure 5.1 indicates this address. In this method, the packet direction through a tunnel is always unidirectional. In contrast to conventional tunnel usage, packets are always encapsulated and sent from end hosts to relay switches (no encapsulated packet is sent from relay switches to end hosts). Therefore, we can use a single IP address that does not exist in the network as the source address for all tunnels. As a result, each relay switch will have only one tunnel between $X.X.X.X$ and the relay switch itself, thereby avoiding the tunnel scalability issue.

Our method, achieving explicit path control by tunneling, can route arbitrary packets through arbitrary paths. It does not restrict wired topologies. Figure 5.2 shows how to specify arbitrary path by using our method. The packet is routed and forwarded in the wired IP network in accordance with the destination of the outer header. When the packet arrives the destination switch, the outer header is removed by the decapsulation process. Finally, the original IP packet arrives the original destination host similar to

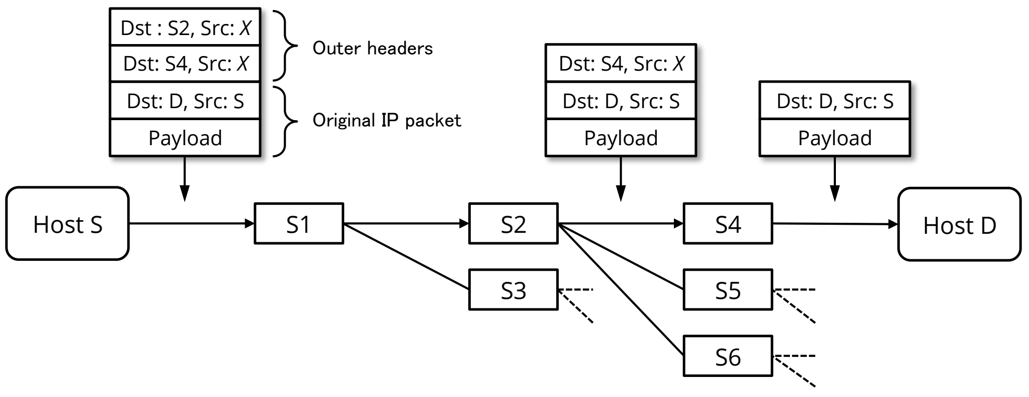


Fig. 5.2. How to specify an arbitrary path via multiple relay switches.

onion routing [81]. Stacking multiple tunnel headers that each header has a relay switch address as a destination enable us to route packets through arbitrary path. This stacking, however, causes further MTU degradation.

5.3.1 Usable Tunneling Protocols

Various tunneling protocols have been proposed, standardized, and developed. The proposed method adopts a tunneling protocol to specify each path similarly to source routing. To achieve tunnel scalability, the source address of outer IP headers from all end hosts are faked as an IP address, as described above. Therefore, those types of stateful protocols that require maintenance of the session between tunnel end points cannot be used. For example, some tunneling protocols using authentication methods such as IPsec technologies and Layer 2 Tunneling Protocols cannot be used for our approach.

In contrast to stateful protocols, stateless protocols can be adopted for our proposed method. Table 5.2 shows usable tunneling protocols. Encapsulation techniques are instrumental for various purposes such as Virtual Private Network (VPN), network virtualization, and software-defined networking. Tunneling protocols have been proposed and developed over time for many uses. Therefore, even if dedicated schemes and hardware for multipathing are not developed, the proposed method can be used throughout actual data center environments. Administrators of data centers should choose among these protocols in accordance with their equipment.

Commodity hardware products have already supported tunneling at a wire speed. For example, VL2 [36] that uses IP-in-IP tunneling constructed a layer-3 network test environment using commodity switch chips Broadcom 56820 and 56514. These chips, which are sometime called *merchant silicon*, support layer-2 and layer-3 packet forwarding, encapsulation and decapsulation at a wire speed. In addition, Microsoft has deployed VL2 into their storage networks [82]. Thus, our method using IP tunneling for multipathing is a credible way for practical commodity-based data center networks.

5.3.2 End Host Modification

The proposed method shifts the functionality of traffic load balancing from the network switches to the end hosts. In this section, we describe the design of the modification for end hosts called iplb. iplb works as a physical network protocol stack after original network protocol stack processing for end-to-end data communication. When a host sends a packet, iplb finds a set of relay switches corresponding to the destination prefix of the

Table 5.2. Usable tunneling protocols for the proposed method.

Protocol name	Specification	Published year
IP in IP Tunneling	RFC1853	1995
GRE	RFC2784	2000
EtherIP	RFC3378	2002
LISP	RFC6830	2011
VXLAN	RFC7348	2014
NVGRE [23]	Internet Draft	2012 ∼
GUE [54]	Internet Draft	2013 ∼
Geneve [26]	Internet Draft	2014 ∼

packet. Then, if the prefix is found, the host selects a path identified by relay switches using some algorithms, which are proper for wired topologies. Finally, the packet is encapsulated in one or more tunnel headers and is transmitted to the wired network. Path selection for fat-tree topologies and random graphs are described in Section 5.4.1 and Section 5.5.1.

iplb is designed as a kernel module. iplb is placed in a packet transmission path in kernel space and it does not need any changes to other communication APIs, socket and packet transmission APIs for instance. iplb maintains a longest prefix match-based table. An entry of the table contains a set of relay switch lists for the corresponding destination network prefix. A list of relay switches identifies a corresponding path from a host to a destination network. A set of the lists identifies paths from a host to a destination network. When applications send packets, iplb checks the destination address of the packet. If the destination address is found in the table, packet is encapsulated in one or more tunnel headers to relay switches selected from the set of relay switch lists for that prefix. In this way, end hosts can split the traffic to destinations via the set of corresponding relay switches.

A naive way for designing and implementing iplb is using existing kernel tunneling APIs and following the network layering model faithfully. This way, however, requires additional protocol processing due to multiple tunnel header stacking as we described in Chapter 4. Thus, we adapt the FIB approach to the design of iplb. In the iplb model, only two lookups are needed to check destination addresses of original packets and select a path for a packet. It is not necessary to route all destination addresses for multiple tunnel headers. Thus, iplb perform only the two lookups and adding tunnel headers without routing lookups for the headers. This design from the overlay FIB contributes to performance improvement due to eliminating additional protocol processing.

We implemented iplb as a kernel module for Linux[*1]. It provides the functionalities described above and an API to modify and control the table in Linux kernel space. This implementation takes out packets from the transmission path using the netfilter hook (`nf_register_hooks()`), adds tunnel headers to the packets and puts them back. The iplb API is designed and implemented as a protocol family in Netlink [62]. It provides operations for prefix insertion and deletion, modifying sets of relay switch lists and dump table entries. Userland application such as control plane systems can configure the iplb table through the API like the following `ip` command examples.

```
Command 1: entry insertion
  $ ip lb add prefix 172.16.0.0/24 relay 10.0.0.1,10.0.0.4 type gre
  $ ip lb add prefix 172.16.0.0/24 relay 10.0.0.1,10.0.0.8 type gre
```

[*1] `https://github.com/upa/iplb`

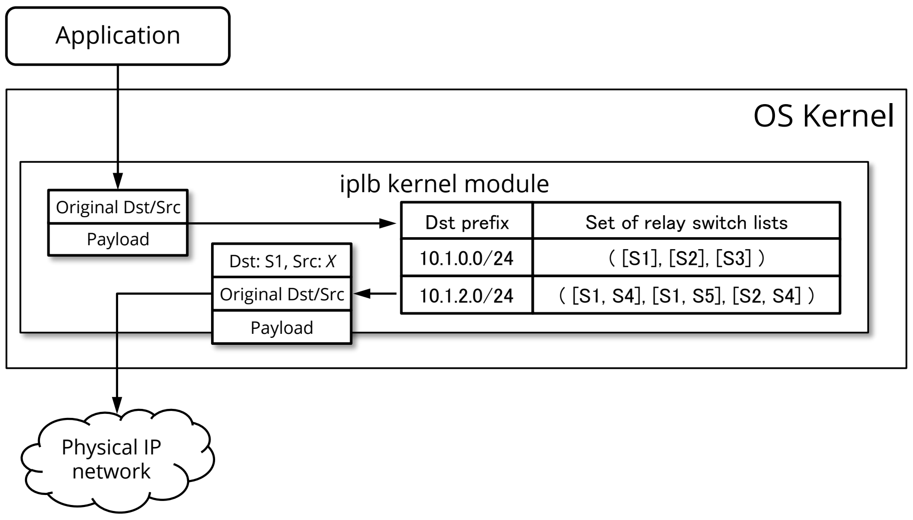


Fig. 5.3. Overview of the modification for software network stack of end hosts.

```
  $ ip lb add prefix 172.16.1.0/24 relay 10.0.1.1,10.0.1.2 type ipip
```

```
Command 2: show table entries
  $ ip lb show
  prefix 172.16.0.0/24 relay 10.0.0.1 10.0.0.4 type gre index 1
  prefix 172.16.0.0/24 relay 10.0.0.1 10.0.0.8 type gre index 2
  prefix 172.16.1.0/24 relay 10.0.1.1 10.0.1.2 type ipip index 1
```

With this configuration, a packet to 172.16.0.0/24 is encapsulated in tunnel headers with destinations 10.0.0.1 and 10.0.0.4 or 10.0.0.1 and 10.0.0.8. Path selection on the fat-tree topologies and random graphs: hash-based, flow-based and round-robin, are described in following sections.

5.4 Adaption to Fat-tree Topology

Next, we adapt iplb for throughput improvement by multipathing in two topologies. In this section, we describe how to adapt iplb to fat-tree topology based layer-3 network and evaluation result on a preliminary experiment with physical equipment and simulation results using an implementation.

We adapt our method for 3-level k-ary fat-tree topology based on [32]. k represents the number of ports of a switch. A 3-level k-ary fat-tree topology has $(k/2)^2$ root switches, k pods, $k/2$ aggregation and edge switches in each pod and $k/2$ end hosts in each edge switch. Thus, it can accommodate $k^3/4$ end hosts. One of most important characteristics of the fat-tree topology is that it has full bisection bandwidth. All switch layers: root, aggregation and edge switches, have same bandwidth for uplinks and downlinks. Therefore, oversubscription does not occur in fat-tree topology from the viewpoint of interconnect bandwidth. In order to achieve full bisection bandwidth in IP network practically, communication traffic between end hosts must be balanced among multiple paths equally.

To balance traffic end hosts transmit among multiple paths, we adapt iplb. In fat-tree topology, one relay switch can specify all shortest path paths. In the case of inner-pod communication, each path between edge switches can be identified by specifying an aggregation switch as a relay switch. In the case of inter-pod communication, each path through aggregation switches can be identified by specifying a root switch. Thus, iplb table

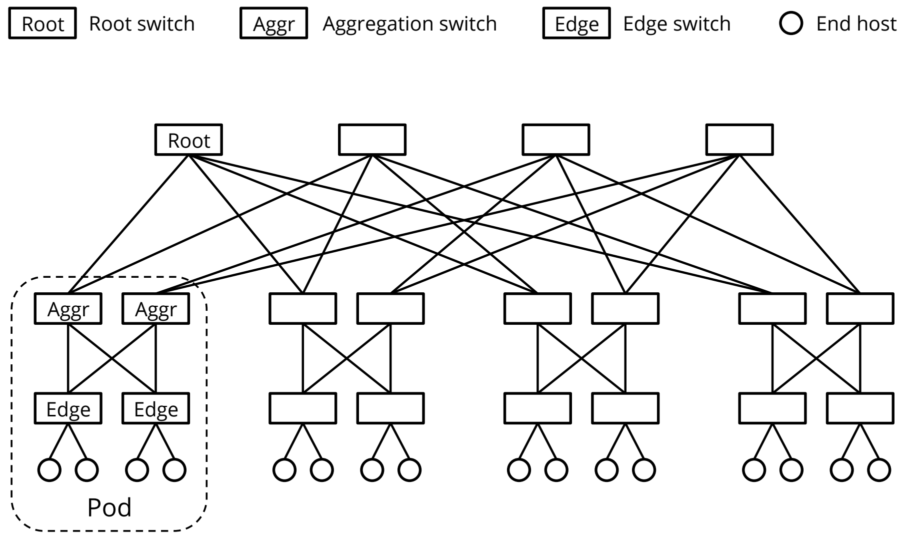


Fig. 5.4. 3-level k-ary fat-tree topology with $k = 4$.

entries for same pod networks have addresses of aggregation switches as relay switches, and entries for other pod networks have addresses of root switches as relay switches.

5.4.1 Flow Assignment Algorithm

In addition to our method of specifying paths, an algorithm for balancing the traffic load across multiple paths is needed. End hosts should split traffic strategically into multiple paths, aiming to maximize the aggregated bandwidth of the whole network. This section describes two algorithms for fat-tree topologies.

Traffic sent from a host consists of flows that are identified by a 5-tuple (source and destination IP addresses, source and destination port numbers, and protocol number). Packets belonging to the same flow should be taken through the same path to avoid packet reordering that would cause performance degradation for both TCP and the applications. Moreover, each flow has different bandwidth. Therefore, the algorithm working in the end hosts should select paths for each flow taking account of the bandwidth of individual flows. In this study, we use two algorithms, called the hash-based and the flow-based algorithms to maximize the aggregated bandwidth.

The **hash-based** algorithm is the same as ECMP. A relay switch for a flow is decided by a calculated hash value from the 5-tuple for the flow. This algorithm does not consider the bandwidth of individual flows, so the traffic will tend to be via a particular path. As a result, it will not be able to maximize the aggregated traffic due to oversubscription as with ECMP.

The **flow-based** algorithm is based on an algorithm proposed by Al-Fares *et al.* [32]. This algorithm considers the bandwidth of individual flows, and attempts to balance traffic equally among multiple paths. Balancing flows among paths equally is a well-known problem and is a variant of the NP-hard bin packing problem [32]. Therefore, Al-Fares *et al.* propose a heuristic algorithm for assigning flows among multiple paths via the modified switch. We adapt this algorithm for use with end hosts. The adapted algorithm is outlined as Algorithm 1.

EncapsulatePacket() is called to decide on a relay switch when packets are sent from applications. Packets belonging to known flows are encapsulated with an outer IP header for the relay switch to which flows are assigned. If a packet is not part of an existing

Algorithm 1 The flow assignment algorithm.

```
function EncapsulatePacket($packet$)
    Hash 5-tuple of $packet$, and find this flow $f$.
    if such a flow $f$ exists then
        update packet counters of flow $f$;
    else
        Record the new flow $f$;
        Assign flow $f$ to a relay by the hash value;
    end if
    Encapsulate $packet$ to relay switch flow $f$ assigned;
end function

function RearrangeFlowForPrefix($prefix$)
    for $relay$ in $relays$ of $prefix$ do
        Find the $Rmax$ and $Rmin$ with the largest and the smallest traffic respectively;
        Calculate $D$, the difference between $Rmax$ and $Rmin$;
    end for
    for $flow$ in $flows$ of $prefix$ do
        Find the largest flow $f$ whose size is smaller than $D$;
    end for
    if such a flow exists then
        Switch the relay point of flow $f$ to $Rmin$;
    end if
end function
```

flow, it is recorded as a new flow and a relay switch is assigned using the hash value of the 5-tuple. *RearrageFlowForPrefix()* is called periodically for each destination prefix. It reassigns the largest flow that is smaller than the margin between the largest and the smallest flows for that destination.

5.4.2 Evaluation

Availability and deployability are achieved by using a commodity-based layer-3 network as described in Section 5.2 and 5.3. In this section, we investigate throughput improvement using our proposed method on the fat-tree topology. To demonstrate that the method can use multiple paths efficiently, we evaluated the method in two cases: 1) as a proof-of-concept for the method, we adopted it for a simple multipath topology, and 2) we evaluated it for a fat-tree topology as a typical example of a data center network.

5.4.2.1 Preliminary Experiment

In the preliminary experiment, we applied the method to a simple multipath topology that had two paths between four hosts. The test topology is shown in Figure 5.5. We prepared the experimental set up using commodity equipment. The switches were Juniper Networks EX4200-48P, the two sender and two receiver hosts were 1U servers with an Intel Xeon L3426 (1.87 GHz) CPU and 4 GB memory. All network interfaces were 1000BASE-T. In the experiment, Sender1 sent test traffic to Receiver1, and Sender2 sent test traffic to receiver 2, balancing the traffic via switch B and switch C using the proposed method. We evaluated the performance in terms of the sum of the received traffic for the two receivers. When not multipathing, the link between switch A and switch B will overload and the overall received traffic will be 1000 Mbps.

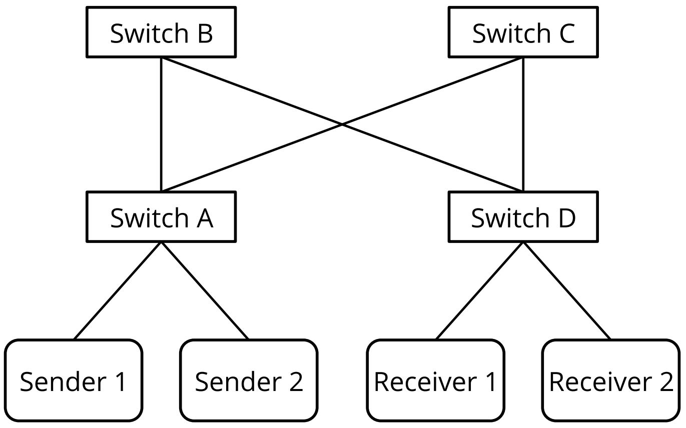


Fig. 5.5. The experimental topology for the preliminary evaluation.

The test traffic from a sender comprised multiple flows. To generate test traffic with multiple flows, we implemented a traffic generator called flowgen*[2] that generates multiple UDP flows to a host. In addition, flowgen can support the following three distribution patterns of bandwidth for individual flows to emulate real traffic.

- Same: the ratio of bandwidth for each flow is uniform.
- Random: the ratio of bandwidth for each flow is random.
- Power-law: the ratio of bandwidth for each flow follows a power law.

In this experiment, all distribution patterns were evaluated with 20 flows. The packet size for the test traffic was 1476 bytes.

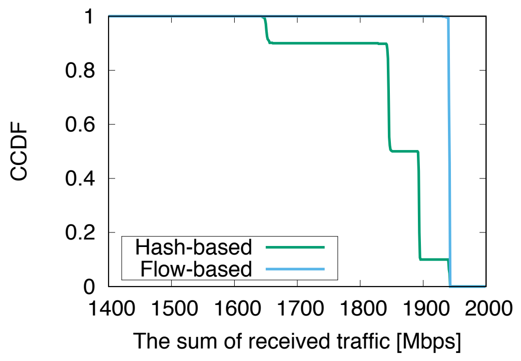
Figure 5.6 shows the results of the experiment. The y-axis represents a complementary cumulative distribution functions for aggregated received traffic on receiver1 and receiver2 for Same, Random, and Power-law flow distribution patterns across 20 runs/permutations of tests for each flow distribution pattern, over 1 minute.

Figure 5.6(a) shows the result for the Same flow distribution pattern. The hash-based algorithm achieves 100% probability up to 1650 Mbps. Further, the traffic trended as expected. In contrast to the hash-based algorithm, the flow-based algorithm achieved 100% link utilization for Same flow distribution pattern. Moreover, for the Random (Figure 5.6(b)) and Power-law (Figure 5.6(c)) patterns, the flow-based algorithm performed better than the hash-based algorithm. This shows that the flow-based algorithm can use multiple paths better than a hash-based algorithm similar to ECMP. However, the flow-based algorithm cannot achieve 100% link utilization for the Random or Power-law flow distribution patterns. This could be caused by the heuristic algorithm and/or possibility that there is no assignment pattern achieving 100% link utilization. In addition, the reason that not all patterns can achieve 2000 Mbps is the encapsulation overhead. This involves 1.5% of throughput reduction with 1476-byte packets in a link. Despite this, for a packet size of 9000 bytes, the throughput reduction is approximately 0.3%.

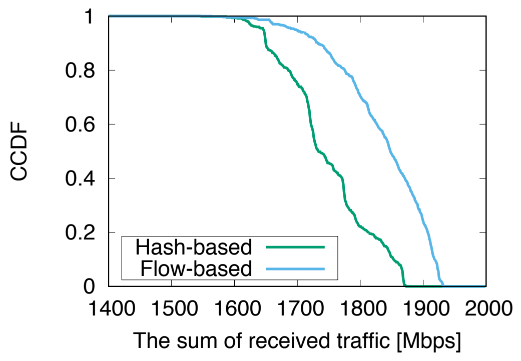
5.4.2.2 Evaluation for a Fat-tree Topology

The preliminary experiment shows the proof-of-concept multipathing using the proposed method. We next evaluated it from the viewpoint of throughput improvement in data center networks. We adopted the method for a 3-level 4-ary fat-tree topology based on the evaluation by Al-Fares *et al.* [32]. The fat-tree topology for this experiment was 3-level

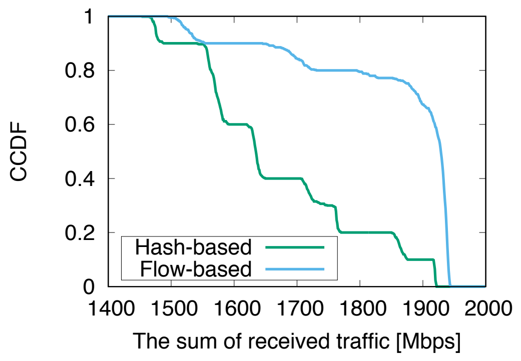
*[2] `https://github.com/upa/flowgen`



(a) Same



(b) Random



(c) Power-law

Fig. 5.6. Results of the preliminary experiments.

k-ary fat-tree topology with $k = 4$ that is the same as Figure 5.4. There are two types of multiple path sets. For inter-pod communication, there are four multiple paths through four root switches. For intra-pod communication, there are two multiple paths through two aggregation switches in each pod.

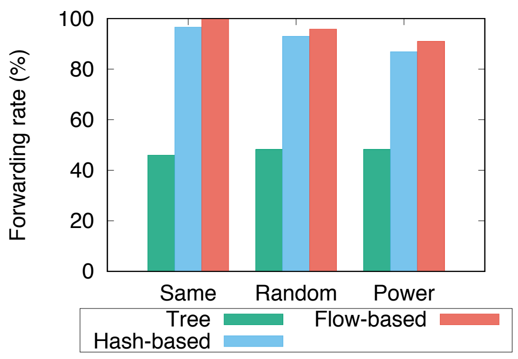
Instead of preparing physical equipment, we simulated the network using ns-3 [83] and Direct Code Execution (ns-3-dce) [84] extension. The ns-3-dce extension can emulate the Linux kernel and its userspace in the ns-3 simulation environment. We emulated the Linux end hosts including the actual iplb implementation that was used in the preliminary experiment. The fat-tree network comprised Linux hosts as layer-3 switches in the ns-3 simulated network.

The test traffic was the same as for the preliminary tests, namely 20 UDP flows and 1024-byte packets, with GRE being used as the tunneling protocol. Distribution patterns of bandwidth for individual flows were also same as the tests, Same, Random and Power-low. The link speed on the simulation environment was 8Mbps in order to reduce simulation running time. All simulation parameters are show in Table 5.3. The mapping of sender and receiver hosts is configured by benchmark suites proposed by Al-Fares *et al.* [32]. Although they proposed five benchmark suites for their dedicated addressing scheme, we used just two of the benchmarks that were not related to a particular addressing scheme, as follows.

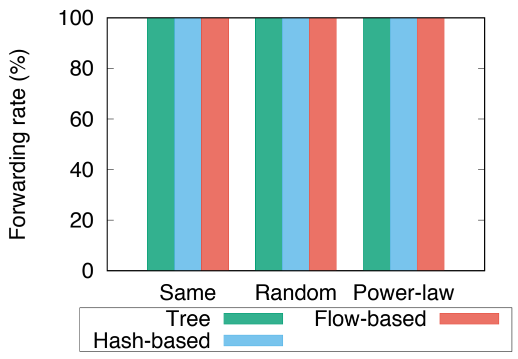
- Random: a host sends to any other host in the network with uniform probability.
- Stride(i): a host with index x will send to the host with index $(x + i) mod 16$.

Table 5.3. The simulation parameters for the fat-tree simulations

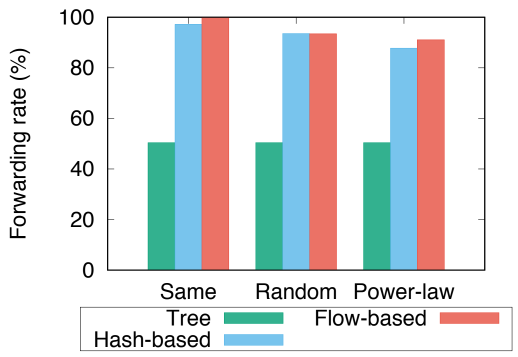
Parameter	Value
Link	Type: Point-to-Point, Speed: 8Mbps, MTU: 1500 byte.
Simulation time	10 sec for configuration, 30 sec for loading test traffic.
Test traffic	20 UDP flows with 1024-byte packets.
Network stack	liblinux [84] (kernel ver 3.17.0).



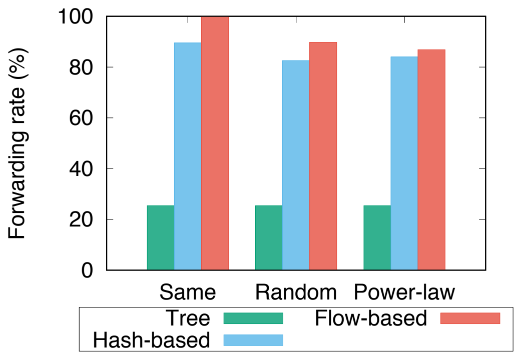
(a) Random



(b) Stride (1)



(c) Stride (2)



(d) Stride (4)

Fig. 5.7. Results of the evaluation for the fat-tree topology.

Figure 5.7 shows the results of the experiments with the fat-tree topology. The forwarding rate is a percentage of the sum of received packets to the sum of sent packets (i.e., the aggregated throughput rate). The results are the averages of 30 runs/permutations of the benchmark tests for each flow distribution pattern. Hash-based and flow-based refer to the balancing algorithms, and tree means there is no multipathing. In the fat-tree topology, there are two paths for communication from two hosts to two hosts in the same pod, and there are four paths for communication from four hosts to four hosts in different pods. Therefore, if the host-to-host mapping is 1-to-1, 100% forwarding rate will be achieved by assigning flows to multiple paths completely equally. As a result, when the flow distribution pattern is Same, the flow-based algorithm achieved 100% forwarding rate for all benchmark suites.

For the Stride(1) benchmark shown in Figure 5.7(b), all traffic is forwarded by edge switches, so there is no overloaded link. For the Stride(2) benchmark shown in Figure 5.7(c), the test traffic is confined to each pod. All test traffic is not routed across root

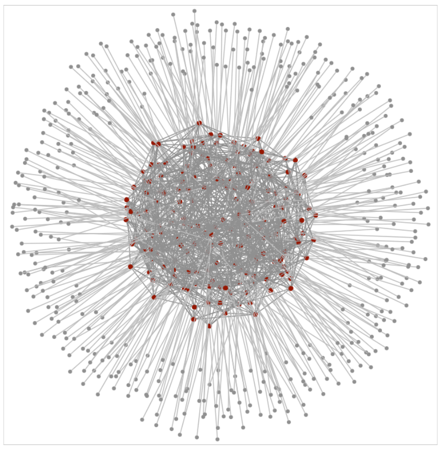


Fig. 5.8. An example of a random graph topology proposed by Jellyfish from [3].

switches. This is similar to four sets of the preliminary test topology, so the results for Stride(2) are the same as the average of those of the preliminary tests shown in Figure 5.6. For the Stride(4) benchmark, all traffic is taken through four root switches. Figure 5.7(d) shows the results for Stride(4). Without multipathing, all traffic goes via one root switch in accordance with the shortest-path tree. Therefore, the forwarding rate for tree is 25%. By contrast, the hash-based and the flow-based algorithms achieved over 80% forwarding rates. Overall, the proposed method using the flow-based algorithm achieved a forwarding rate of over 85% for all benchmark and flow distribution patterns.

From the results, our proposed method could use multiple paths efficiently, with a forwarding rate of over 85% for a fat-tree topology. We demonstrated that the method can enhance availability by enabling a layer-3 network design and improve throughput by multipathing for data center networks.

5.5 Adaption to Random Graph

In this section, we describe how to adapt iplb to layer-3 random graph networks to demonstrate that iplb can be adapted to other data center topologies. Jellyfish [33] has proposed random graphs as the topology for data center networks. Figure 5.8 shows an example visualization of a jellyfish topology. In order to utilize a lot of links of a random graph for throughput improvement, Jellyfish proposes using k-shortest path routing and MultiPath TCP. However, commodity switches supporting only shortest path forwarding in their hardware cannot achieve k-shortest path routing. We therefore adapt iplb to achieve this. In addition, we propose a control plane system that dynamically configures iplb tables on end hosts in random graphs. This control plane system only requires OSPF, which is the most popular commodity routing protocol.

5.5.1 k-shortest path routing with MPTCP

A random graph of Jellyfish is defined as follows: Each switch i has some number k_i of ports, of which it uses r_i ports to connect to other switches, and uses the remaining $k_i - r_i$ ports for end hosts. r_i switches that a switch connects to are selected with uniform probability. In this study, we also consider by default as with the Jellyfish paper, every switch has the same number of ports and servers: k ports for servers and r ports for other switches.

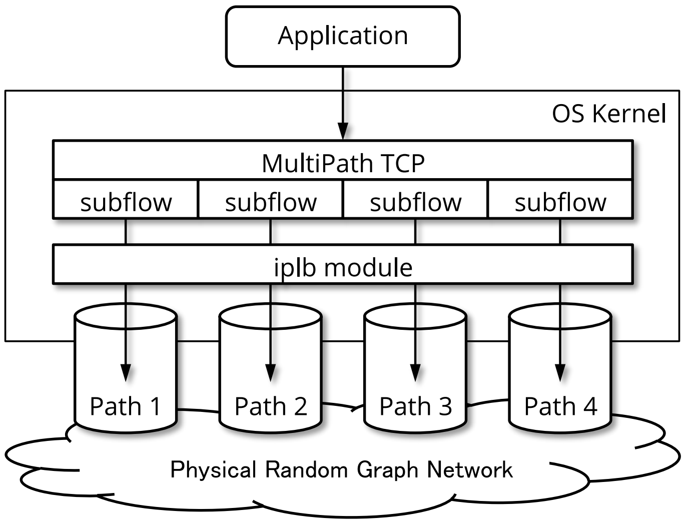
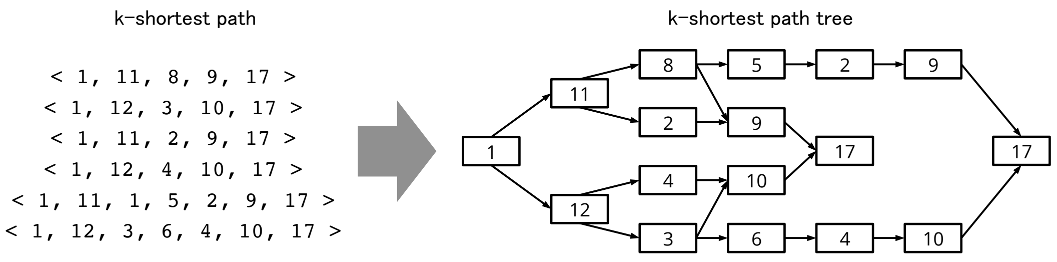


Fig. 5.9. Combination of MPTCP and iplb on an end host.

k-shortest path is some number k paths from a source node to a destination node in a graph. Yen's algorithm [85] finds the shortest path and $k-1$ other paths in nondecreasing order of cost. In contrast to ECMP that utilizes only same cost shortest paths, k-shortest paths can utilize more links. To split traffic from an end host into multiple paths, Jellyfish uses MultiPath TCP (MPTCP) that is a transport layer protocol. An MPTCP connection is composed of multiple TCP connections called subflows. Moreover, MPTCP has a congestion control system to manage multiple connections [86]. Thus, MPTCP achieves throughput improvement of connections, connection migration, and robustness by assigning subflows to individual links or IP addresses. Jellyfish advocates assigning MPTCP subflows to k-shortest paths to improve aggregate throughput of the whole network.

Commodity switch products, however, do not support k-shortest path routing as a result of commoditization of and integration into IP. Therefore, we adapt iplb to specify k-shortest paths in commodity-based random graph networks. Figure 5.9 shows the combination of MPTCP, k-shortest path and iplb. The iplb table contains k-shortest paths as a set of relay switch lists for each destination. When an application sends data, the MPTCP at the transport layer splits it into multiple subflows. Next, iplb assigns subflows into individual paths to the destination of this communication. A series of packets forming a subflow are encapsulated in tunnel headers to relay switches that represent a k-shortest path. Assigning paths for individual subflows are decided in a round-robin fashion. iplb does not need to consider flow bandwidth in contrast to the fat-tree case described in Section 5.4 because MPTCP performs congestion control across multiple paths.

Ordinary use cases of MPTCP require multiple network interface cards (NICs) for assigning subflows to multiple paths via different NICs. By contrast, the iplb can use multiple paths via one NIC. iplb distinguishes a subflow from others by 5-tuple; therefore, multiple subflows having different source port numbers via one NIC can also be distinguished and assigned to different paths. Creating subflows with different port numbers over one NIC depends on MPTCP implementation. Fortunately, Linux MPTCP implementation has this function as the `ndiffports` path manager [87] because this path creation is known as an efficient technique for data center networks [88].



```
< 1, 11, 8, 9, 17 >
< 1, 12, 3, 10, 17 >
< 1, 11, 2, 9, 17 >
< 1, 12, 4, 10, 17 >
< 1, 11, 1, 5, 2, 9, 17 >
< 1, 12, 3, 6, 4, 10, 17 >
```

Fig. 5.10. An example of making a tree from k-shortest path with $k = 6$.

5.5.2 A control plane system using OSPF

iplb is just a technique to achieve explicit path control; therefore, we have to find relay switches that form k-shortest paths to take packets through the paths. This section describes how to find relay switches to identify k-shortest path and a control plane system that finds the relay switches and dynamically configures end hosts for failure recovery.

k-shortest paths can transform into a tree called a k-shortest path tree, which has a source switch to a destination switch. Figure 5.10 shows an example of making a k-shortest path tree from lists of k-shortest paths. Figure 5.10 depicts k paths from a source switch 1 to destination switch 17 and a k-shortest path tree constructed from them in an example random graph, in which each switch has unique number. Identical switches that appear at different hops in different paths are inserted as different nodes for eliminating loops in the tree. To find relay switch lists representing each path of k-shortest paths, we pick out nodes after branch nodes.

A simple variant of the depth-first search shown in Algorithm 2 finds relay switch lists from a k-shortest path tree. The algorithm searches a tree starting from a source and pushes nodes where parent nodes are branch nodes; they have two or more leaf nodes. When reaching the destination node, it dumps the *relaylist* that represents the relay switches of the path of the k-shortest paths and returns to the last branch node. For example, in the case of Figure 5.10, the algorithm processes node 1 and 11, then it pushes node 11 to the *relaylist* because parent node 1 has two leaf nodes. Next, node 2 is also pushed to the *relaylist* because node 11 is a branch node. Node 9 is not pushed to the list because the parent is node 2 in this search and is not a branch node. Finally, it reaches the destination node 17 and dumps the relaylist $\langle 11, 2\rangle$ that represents the relay switch list of the path $\langle 1, 11, 2, 9, 17\rangle$. As a result, we can find that the relay switch list for the path $\langle 1, 11, 8, 5, 2, 9, 17\rangle$ is $\langle 11, 8, 5\rangle$ and the list for the path $\langle 1, 11, 8, 9, 17\rangle$ is $\langle 11, 8, 9\rangle$ in the same manner. Thus, we find relay switch lists for k-shortest paths.

We simultaneously propose a control plane system, called iplbd, to dynamically configure relay switch lists, although the iplb tables can be configured manually and statically. In actual data center networks, the topology may change due to a variety of reasons: maintenance, enhancements and breakdowns. Therefore, an automatic configuration, failure detection and recovery mechanisms are needed for high availability.

We use OSPF [72], which is a routing protocol commonly implemented in switch products. OSPF nodes distribute Link State Advertisement (LSA) packets that contain link statuses of the sender nodes to the whole OSPF network. By flooding LSAs, OSPF nodes know the complete graph information about the network. This graph information is named Link State Database (LSDB). When link or node failure occurs, new LSA is flooded and all OSPF nodes update their LSDBs. Thus, completeness of the LSDB is guaranteed by the protocol design. OSPF and LSDB is used to calculate the shortest

Algorithm 2 Gathering relay switch lists from a k-shortest path tree.

```
function KtreeDfsearch($V$, $relaylist$, $parent$)
    if $V.leaflist$ is empty then
        $V$ is destination node. dump $relaylist$.
        return.
    end if
    if $parent.leaflist > 1$ then
        push $V$ to $relaylist$.
    end if
    for $v$ in $V.leaflist$ do
        KtreeDfsearch($v$, $relaylist$, $V$)
    end for
end function
```

paths on each node for IP routing and forwarding. Instead, we use OSPF to calculate k-shortest paths and relay switch lists.

iplbd works as a userland process at end hosts. iplbd is composed of two parts, 1) OSPF protocol processing and 2) configuring the iplb table in the kernel space. iplbd establishes the OSPF neighbor connection with the switch accommodating the end host, on which the iplbd process runs, and constructs LSDB receiving LSA packets. When LSDB construction is finished, iplbd calculates the k-shortest path and relay switch lists to all switches and installs the result to the iplb table via the API described in Section 5.3.2. When switch of link failures occur, new LSA is flooded and LSDBs on end hosts are updated, then all relay switch lists of all hosts are updated autonomously. Failure recovery and availability are achieved in this manner.

The scalability of OSPF depends on the number of nodes; therefore, it may not be feasible to make end hosts speak OSPF. Some north-bound APIs can avoid this infeasibility. The north-bound API is a general term of protocols that provide configuration of acquisition interfaces for control plane systems running on devices or controllers. The BGP-based Link-State distribution (BGP-LS) [89] is one of the north-bound APIs. OSPF nodes can distribute LSDB information to other applications through BGP sessions. iplbd can collect complete graph information without OSPF processing by using BGP-LS for instance. Quagga [90], one of the most famous open source routing software, also has such a north-bound API, called OSPF-API. In addition, collecting graph information using OSPF or north-bound APIs, and calculating paths are feasible methods for not only random graphs, but can also be applied to fat-tree topologies described in Section 5.4 and further path computation techniques.

5.5.3 Evaluation

In this section, we investigate throughput improvement on random graphs in accordance with Jellyfish. Our method improves aggregated throughput with only commodity layer-3 switches by utilizing more links than ECMP. To demonstrate this, 1) we evaluated three random graph topologies with the iplb on a simulation environment. Additionally, we investigated 2) the overhead of multiple tunnel headers and 3) the time to recover link failure with iplbd.

5.5.3.1 Throughput improvement

First, we simulated three random graph networks using ns-3-dce [84]. Basic configurations are the same as the experiment on the fat-tree in Section 5.4.2.2. The random graph

Table 5.4. The simulation parameters for the random graph simulations

Parameter	Value
Link	Type: Point-to-Point, Speed: 10Mbps, MTU: 9000 byte.
Simulation time	10 sec for configuration, 30 sec for loading test traffic.
Test traffic	1 TCP flow split into 8 subflows by MPTCP.
MPTCP path manager	`ndiffports` (8 subflows).
TCP congestion control	CUBIC (default on Linux)
Network stack	liblinux [84] (kernel ver 3.13.33, MPTCP ver 0.89 [87]).

Table 5.5. Topology sizes of simulated random graph networks

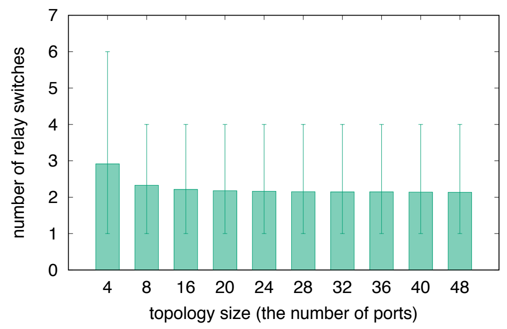
# ports of a switch	# switches	# end hosts
4 ports	20	16
6 ports	45	54
8 ports	80	128

Table 5.6. The simulation results for random graph networks

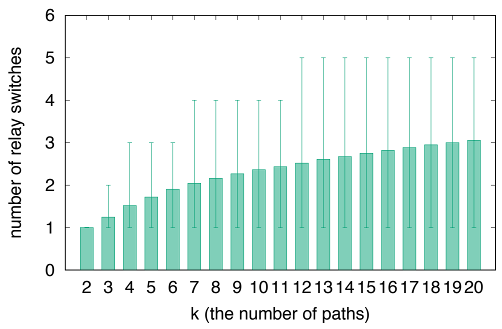
Topology size	ECMP	k-shortest path (k)						
		2	3	4	5	6	7	8
4 ports	1.12	1.18	1.22	1.23	1.22	1.23	1.19	1.17
6 ports	1.21	1.19	1.28	1.30	1.28	1.30	1.30	1.31
8 ports	1.25	1.16	1.28	1.31	1.31	1.31	1.30	1.30

networks were composed of Linux hosts as layer-3 switches in the ns-3 simulated network. In contrast, the end host Linux kernel was the Linux MPTCP version that is an out-of-tree implementation of Linux [87], and test traffic was TCP. All simulation parameters are shown in Table 5.4. In this simulator, we simulated three random graph networks having different numbers of switch ports. The numbers of ports and switches are equivalent to 3-level k-ary fat-tree topologies. Table 5.5 shows the numbers of ports, switches and end hosts on each test topology. We used GRE for iplb encapsulation. During the first 10 seconds in the simulations, the basic IP shortest path routing table and the iplb table for k-shortest path on all end hosts were configured. During next 30 seconds, test traffic was loaded. The communication pair in this test was random: a host sends to any other host in the network with uniform probability.

Table 5.6 shows the results of the averages across 15 runs per topologies. The values indicate normalized aggregated received bytes on the basis of aggregated received bytes on the shortest path routing without ECMP and iplb. From the results, in all cases, our method achieved higher throughput than shortest path routing. In the 4 ports switch topology, k-shortest path routing with iplb achieved higher throughput than the ECMP in all cases. Additionally, in the 8 ports switch topology, it achieved 1.3 times aggregated throughput for the shortest path routing. These results are also better than ECMP and mean k-shortest path routing and iplb can utilize more links than ECMP on random graphs. In contrast, if k was small ($k = 2$), the throughput was lower than ECMP because a too small k cannot make full use of links. Overall, the simulation results show that our method achieves a throughput improvement on random graphs in accordance with jellyfish, on commodity-based layer-3 random graph networks using the encapsulation technique.



(a) The number of relay switches on different size topologies with $k = 8$



(b) The number of relay switches with k on 16 ports topology

Fig. 5.11. The number of relay switches on random graphs.

5.5.3.2 Overhead of multiple tunnel headers

iplb uses multiple tunnel headers for a packet to realize explicit path control in layer-3 networks. The payload size of packets decreases with the increase in the number of relay switches for a path. Payload size reduction causes throughput degradation. Thus, we investigated the number of relay switches on different size random graphs and clarified the throughput degradation due to multiple tunnel headers. In addition, we measured the packet transmission throughput of an end host with/without the iplb kernel module because it may also be a factor of degradation at the end host TX path.

In order to investigate the number of relay switches depending on topology sizes and k of k-shortest paths, we implemented a simple simulator that generates a random graph and calculates k-shortest paths and relay switch lists from end hosts to other end hosts. Figure 5.11(a) shows the number of relay switches required to specify k-shortest path to other hosts with $k = 8$. The x-axis indicates topology sizes and the y-axis indicates the average, maximum and minimum numbers of relay switches for k-shortest paths. From this figure, in the case of most small topology size (4 ports, 16 end hosts and 20 switches), many relay switches were required to specify a path. This is because the topology size is too small to make 8 k-shortest paths; therefore, many relay switches are needed. In the topologies with over 8 ports switches, the average was two and the maximum was four relay switches regardless of topology sizes.

When using GRE format for encapsulation, one tunnel header reduces the payload size of a packet by 24 bytes. When two tunnel headers are required to specify a path, the payload size is reduced by 48 bytes. The 48-byte overhead of the packet size is 3.2% throughput degradation with 1500-byte MTU networks. However, jumbo frames (9000-byte MTU) can be applied to data center networks because a data center network is operated by an individual operator such as cloud or content service providers. With a 9000-byte MTU, the 48-byte overhead causes just 0.5% throughput degradation. In addition, the degradation rate due to four tunnel headers that is the maximum number of relay switches on over 8 port size topologies is just 1%. Thus, the overhead due to multiple tunnel headers is not a problem on data center networks that can use jumbo frames.

Figure 5.11(b) shows the number of relay switches to specify different numbers of k shortest paths. The y-axis indicates the average, maximum and minimum numbers of relay switches for k-shortest paths similar to Figure 5.11(a). From this figure, the number of relay switches increases with k. Even if $k = 20$, the number of necessary relay switches is

Table 5.7. Throughput degradation due to the iplb kernel module (Mbps).

Link Speed	without iplb	standard deviation	with iplb	standard deviation	decreasing rate
1-Gbps	989.6	0.10	987	1.38	0.30%
10-Gbps	8929	38.5	8925	104.1	0.04%

three on average. Furthermore, the simulation result on Section 5.5.3.1 demonstrated that $k = 8$ is sufficient to achieve throughput improvement using k-shortest path and MPTCP on random graphs as the Jellyfish paper advocated. Thus, this result also indicates that the number of relay switches that is the number of tunnel headers is not a problem on practical data center networks.

In addition, we measured transmitting throughput with and without the iplb kernel module on Linux. We prepared two Linux hosts and a layer-3 switch in a line topology, in which both hosts were connected to the switch directly. A host sent test traffic via iplb, and the layer-3 switch decapsulated the test traffic and forwarded them to the opposite host. We conducted this test on two cases with 1-Gbps interconnect and 10-Gbps interconnect. In the 1-Gbps test case, the sender and receiver hosts had Intel Xeon L3426 1.87GHz CPU, 4GB memory and Intel e1000 82574L 1-Gbps NICs. The layer-3 switch was Juniper Networks EX4200. In the 10-Gbps test case, both hosts had Intel Xeon CPU E5-2650 2.60GHz CPU, 32GB memory and Intel X520 82599ES NICs. The layer-3 switch was Juniper Networks QFX5100. Link MTU in all experiments was 9216-byte that is the maximum MTU size of Juniper Networks switches. The sender host sent single TCP traffic during 10 seconds using iperf.

Table 5.7 shows the average of 20 runs per permutation of these experiments. The decreasing rate refers to the rate of transmitting throughput with iplb to without iplb. One encapsulation using GRE causes 24-byte packet size reduction, that means 0.26% throughput reduction on the 9216-byte MTU size. The measured value in 1-Gbps links shows that the decreasing rate of throughput is 0.3%. This result is mostly in agreement with the theoretical value of the payload size reduction. Further, in 10-Gbps links, the rate was 0.04%; however, it falls within an error range. From the results, with both link speeds, throughput degradation due to the iplb kernel module including longest prefix match and encapsulation is not a bottleneck in the end host packet TX path.

5.5.3.3 Fault tolerance

Our method uses OSPF for topology detection, failure recovery and calculating k-shortest path. iplbd processes running on end hosts detect the network topology and calculate k-shortest path based on the complete graph information of LSDB. OSPF is a matured technology, whose reliability and deployability are proved by a large number of running IP networks. VL2 [36] also uses OSPF for their IP network, and many enhancements to adapt OSPF to larger networks have been proposed [91]. Although end hosts can obtain topology information by using OSPF, the calculation time for k-shortest paths has to be considered. The computational complexity of Yen's algorithm is $O(kn(e + nlogn))$ where e is the number of links and n is the number of switches. Each end host calculates paths to each switch that accommodates end hosts. When topology change happens, all end hosts must calculate paths for each switch. In order to demonstrate that the calculation is done in practical time, we measured the time that iplbd calculates k-shortest path. In addition, we demonstrate that failure recovery is achieved on a physical network by iplbd.

First, we measured the calculation time for k-shortest paths. In this experiment, we inputted test topologies generated randomly to iplbd, and measured the calculation time

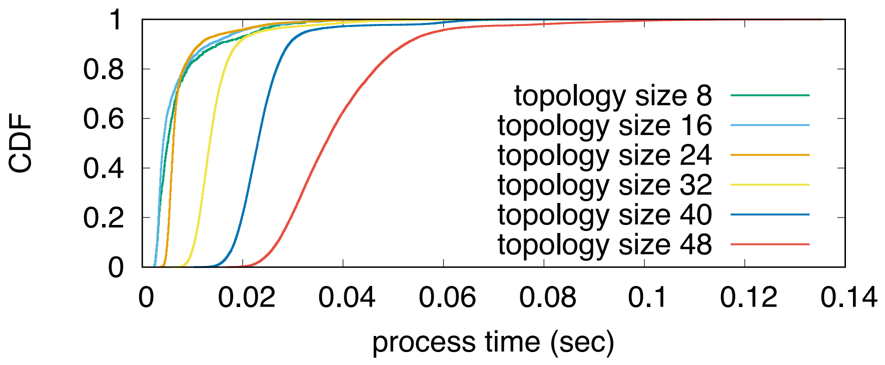


Fig. 5.12. The calculation time for k-shortest path to a destination.

for k-shortest paths to a destination with $k = 8$. The machine used for the experiment had an Intel i7-3770K 3.50GHz CPU and 32GB memory, and PyPy 2.0.2 (Python 2.7.3) was used to run the iplbd implementation. Figure 5.12 shows the experiment result. The y-axis represents a cumulative distribution function across 10 runs/permutations of different size topologies. The topology sizes, the number of ports and switches, were also equivalent to the experiment in Section 5.5.3.1 and 3-level k-ary fat-tree topologies.

From the result, when topology size was 8, 16, or 24 ports, 90% of the calculation times for k-shortest paths for a single destination fell within 0.01 seconds. With 24 ports switches, the number of end hosts is 3456 and the number of switches is 720. The sum of calculation times to all switches is approximately 7.2 seconds. With 24 ports switches, 90% of the calculation times fell within 0.02 seconds; however, calculating paths to all switches requires approximately 32.4 seconds. Therefore, the largest size that can be calculated within a practical time by the current implementation is a random graph constructed from 16 ports switches.

To reduce the calculation time, we plan to implement more optimized iplb and faster k-shortest path algorithms. For example, Hershberger *et al.* have proposed an algorithm, whose computational complexity is less than Yen's algorithm [92].

Next, we conducted an experiment to demonstrate failure recovery: detecting failures and reconfiguring paths by iplbd using a minimum physical environment. The topology for this experiment is shown in Figure 5.13. We prepared two hosts and two paths between the hosts using four switches. OSPF ran on all switches and iplbd ran on both hosts. After the configuring paths by iplbd, the sender host sent TCP test traffic using iperf. Traffic was split into 8 subflows by MPTCP, and each subflow was transferred to the receiver host through two paths via switch B and switch C. Then, we removed switch C from the network by disabling links, and measured received traffic bandwidth on the receiver host every 0.5 seconds.

Figure 5.14 shows the transition of received bandwidth during the experiment. The switch C was removed at 48 seconds, and then the bandwidth decreased to about 600Mbps. After that, bandwidth returned to about 900Mbps after 1.5 seconds. When switch C was removed, switch A and D detected the removal via link down and sent new LSA, then iplbd at the sender host updated LSDB and calculated new paths to the receiver host. The path through switch B kept the communication during the removal, and bandwidth was recovered after re-configuration by iplbd. The subflow through the removed path was re-assigned to another path. Overall, we demonstrated that our method achieves automatic failure recovery and availability using the OSPF-based control plane system in commodity-based layer-3 networks.

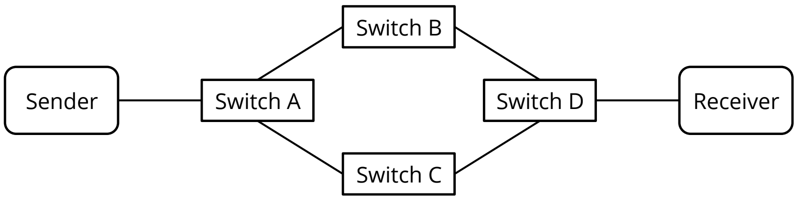


Fig. 5.13. The experimental topology for failure recovery.

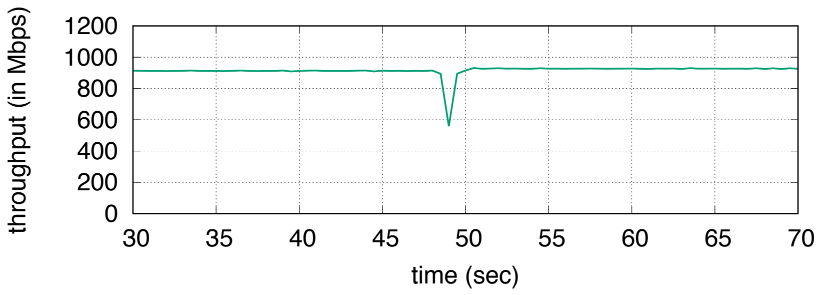


Fig. 5.14. A transition of received bandwidth.

5.6 Summary

We have proposed an explicit path control method for multipathing in layer-3 networks with unmodified commodity switches via the novel usage of common tunneling protocols supported by commodity hardware. End hosts encapsulate packets in one or more tunnel headers, and the packets are taken through explicit paths represented by specified relay switches. End hosts manage path status and traffic balancing so that any modifications for network switches are not needed. We designed and implemented the required modification of end-host software. We demonstrated that our proposed method could be adapted to fat-tree topologies and random graphs. In fat-tree topologies, our method could use multiple paths efficiently, with a forwarding rate of over 85%. In random graphs, out method achieved better aggregate throughput than shortest path forwarding and ECMP with only commodity switches. Additionally, we showed that encapsulating packet in multiple tunnel headers does not degrade throughput considerably. Our control plane system using OSPF provided feasible failure recovery. As a result, the method could enhance availability by enabling a layer-3 network design and improve throughput by multipathing for data center networks.

The main contribution of this research is that explicit path control achieves efficient link utilization with low-end commodity network devices. The explicit path control overcomes inefficient link utilization due to shortest path forwarding. The flow assignment algorithm and MPTCP-based congestion control shown in this research are dedicated methods for fat-tree and random graph topologies. However, iplb is just a technique to achieve path control, and it can be combined with other path computation for other topologies and flow optimization techniques [32, 93, 94, 95]. Furthermore, shifting functions for path management and traffic balancing to end host software network stack makes network nodes simple. This design follows the End-to-End principle: it is better to implement such complex functions at end hosts rather than in intermediate nodes [12]. As a result, the design enables us to continue to use existing commodity IP network devices for practical data center network construction.

From the viewpoint of the virtual network architecture, this chapter also presented a method to exploit the potential for optimization. This method is based on the principle that physical networks for the locator aspect are separated from data communication context, so that multiple locators for path control can be added to packets without disrupting end-to-end data communication. The original network protocol stack of end hosts performs the identifier aspect for end-to-end data communications. Meanwhile, the iplb module performs an optimized physical network protocol stack for locator networks to achieve multipathing by explicit path control. The iplb takes traffic through arbitrary paths by adding multiple locators to a packet in accordance with the 5-tuple of flow as an identifier. The iplb achieves this improvement by enhancing functions of tunneling: path selections as determining destination based on flows, and adding multiple outer headers simultaneously as encapsulation. The significance of this research is that we have solved the inefficient link utilization problem due to shortest path forwarding based on the separation of identifier and locator aspects.

Chapter 6

Conclusion

The commoditization of Internet technologies has caused both positive and negative effects. IP has become the integral and fundamental component of the current data communications, therefore, we have to use IP as is without any modifications to continue to use existing applications and inexpensive commodity network devices. However, IP is insufficient for current requirements, which have been diversified by emerging applications such as cloud computing. Domain separation is not supported under IP because it was originally designed for a single, physical and scalable network. Furthermore, improving aggregate network throughput is difficult in data center environments because of shortest path forwarding and its inefficient link utilization. Our observation is that we can neither discard nor modify IP, though the existing IP-based networks could not satisfy all the emerging requirements.

In this dissertation, we focused on the tunneling approach to improve IP networking because tunneling makes it possible to continue to use existing IP-related protocols and network devices without any modifications. Tunneling allows us to add functionality for domain separation; however, its extensibility is inadequate because the packet transmission performance at end hosts degrades due to additional protocol processing. Moreover, tunneling is an end-to-end technique, so that the inefficient link utilization problem at networks still remains. This dissertation aimed to achieve extensibility with performance improvement at both end hosts and networks through improving the tunneling approach.

In Chapter 3, we introduced a new architectural view for tunnel-based virtual networking and its potential for optimization to overcome the drawbacks of tunneling: performance degradation at end hosts and inefficient link utilization at networks. First, we advocated that tunneling is a boundary between virtual and physical networks through the exploration of details of tunneling protocols. Next, in the architectural view, we presented that the identifier aspect used for data communication and the locator aspect for packet transport are separated into virtual and physical network protocol stacks. Tunnels act as the boundary between virtual and physical networks. This view changes the interpretation of outer networks of tunnels to simple packet transport. The physical network protocol stack for simple packet transport can be optimized preserving end-to-end data communications in the separated virtual network protocol stack. Some implementations accidentally take the form of the architecture; however, they do not exploit this potential for optimization. VM and container technologies also separate network protocol stacks into individual instances. Nevertheless, host OSes use the same network protocol stack implementation for both virtual and physical networks due to the current design. Therefore, existing implementations do not exploit the potential that the physical network protocol stack can be optimized.

In Chapter 4, we proposed the first implementation method that exploits the potential for optimization to improve performance with tunneling at end hosts. Tunneling involves performance degradation due to the additional protocol processing required compared to no encapsulation. We, therefore, proposed the overlay FIB architecture for virtual networking based on the bottleneck analysis of tunneling protocols. The overlay FIB provides a shortcut for physical network protocol processing after tunneling. The FIB approach, which crosses layer boundaries, shows that lookups on each layer can be merged into one lookup. Hence, the overlay FIB offsets the transmission throughput degradation due to the additional protocol processing by avoiding lookups. This technique is based on the fact that the locator aspect is separated from the data communication context; consequently, the physical network protocol stack can shrink during optimization. The overlay FIB achieves this optimization by enhancing the tunneling functions: using a protocol-independent lookup method in the tunnel to determine the destination and simultaneously adding lower-layer headers as encapsulation. The evaluation results showed that the method decreased the CPU time required to transmit a packet by over 12% in general, and the kernel throughput was approximately doubled in particular tunneling protocols. The method demonstrates that the separation of the identifier and the locator can solve the performance degradation due to tunneling.

In Chapter 5, we proposed the second implementation method that exploits the potential for optimization to improve performance at networks through multipathing. The proposed method, iplb, is an alternative physical network protocol stack to achieve explicit path control in commodity-based layer-3 data center networks. The end hosts add multiple locators as outer headers to packets to identify paths, and the packets are routed through the explicit paths represented by the specified locators. By balancing traffic among multiple paths, the aggregate throughput of a network was improved. Shifting the management functions for path control and traffic balancing from networks to end hosts make networks simpler, allowing multipathing to be achieved using only commodity network devices. We adapted this method to fat-tree and random graph topologies, in which the method achieved better aggregate throughput than shortest path forwarding by using multiple paths efficiently. In addition, the control plane system using OSPF could provide feasible availability. This technique is also based on the separation of the locator and identifier aspects. Multiple locators for path control can be added to packets without disrupting data communication. The iplb achieves this improvement by enhancing the tunneling functions: path selections determining the destination based on flows, and multiple tunneling headers added simultaneously as encapsulation. The method demonstrates that separating the identifier and the locator can solve inefficient link utilization due to shortest path forwarding using only commodity network devices.

As discussed above, this dissertation improved IP networking without modifications to IP-related protocols and devices by introducing the architectural view for tunnel-based virtual networking. This view that separates identifier and locator aspects into individual protocol stacks brings the potential for optimization. Furthermore, adopting the tunneling approach provides reusability of existing protocols and devices. Two methods, the overlay FIB and iplb, demonstrated exploitation of the potential by optimizing physical network protocol stack behavior. The optimization relieved the performance degradation due to tunneling at the end hosts, and the stacking multiple locators achieved multipathing for aggregate throughput improvement in IP networks. Then, we have improved IP networking without making any modifications to IP itself. The significance of our approaches in this dissertation is that they improve networking performances without requiring any modifications to existing protocols and network devices, and consequently, they can be deployed to current networked systems easily. Existing applications can still work on virtual networks, and commodity network devices can still be utilized for scalable physical networks. Deploying such dirty-slate approaches requires less time, effort, and cost than replacing existing architecture with clean slate architecture.

The view of this dissertation, which separates two aspects, is applicable to design new clean-slate network architectures. Based on layering approaches, each layer should be responsible for individual functions avoiding duplication for simplicity. From this perspective, when designing clean-slate network architectures or protocols, locator aspects for packet transport and identifier aspects for data communication should be separated functions, protocols, and implementations because they are different aspects of networking. Designing both aspects individually provides the potential for optimizing protocols and implementations for each aspect. Moreover, this separation provides practical deployability. If a more scalable or efficient routing protocol is proposed, only the network layer protocol in the protocol stack for the locator aspect should be replaced, and identifiers for end-to-end communication can remain for practical deployment. Furthermore, merging lookups crossing layer boundaries and the end host-driven explicit path control will be able to help in improving performance when network protocols change even in the future.

Publication and Research Activities

- **Journals**
  1. 中村 遼，石原知洋，関谷勇司，江崎 浩, “トンネリング技術を用いた汎用機器によるデータセンターマルチパスの実現”, 日本ソフトウェア科学会, コンピュータ・ソフトウェア, ネットワーク特集号, Vol.33 (2016), No.3, pp. 29-43, 2016 年, DOI:10.11309/jssst.33.3_29
  2. Ryo Nakamura, Kouji Okada, Yuji Sekiya, and Hiroshi Esaki : “A common data plane for multiple overlay networks”, Elsevier, Computer Networks Journal, ISSN 1389-1286, October 2015, DOI:doi:10.1016/j.comnet.2015.09.031
  3. 山本 成一, 中村 遼, 上野 幸杜, 堀場 勝広, 関谷 勇司: “GINEW : 革新的なネットワーク運用管理アーキテクチャの一提案”, 日本ソフトウェア科学会, コンピュータ・ソフトウェア, Vol.32 (2015), No.3, pp. 4657, 2015 年, DOI:10.11309/jssst.32.3_46
- **International Conferences**
  4. Ryo Nakamura, Yohei Kuga, Yuji Sekiya and Hiroshi Esaki, “Protocol-Independent FIB Architecture for Network Overlays”, ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2016), Hong Kong, China, August 2016
  5. Ryo Nakamura, Yohei Kuga, Yuji Sekiya and Hiroshi Esaki, “Protocol Independent NIC Offloading for Overlay Networks”, ACM CoNEXT 2015 Student workshop, Heidelberg, Germany, December 2015
  6. Ryo Nakamura, Kazuya Okada, Shuichi Saito, Hiroyuki Tanahashi and Yuji Sekiya, “FlowFall: A Service Chaining Architecture with Commodity Technologies”, 2nd IEEE International workshop on CoolSDN 2015 (in conjunction with ICNP 2015), San Francisco, USA, November 2015
  7. Ryo Nakamura, Yuji Sekiya and Hiroshi Esaki, “Layer-3 Multipathing in Commodity-based Data Center Networks”, 18th IEEE Global Internet Symposium (GI 2015), Hong Kong, China, April 2015
  8. Hajime Tazaki, Ryo Nakamura and Yuji Sekiya, “Library Operating System with Mainline Linux Network Stack”, The Technical Conference on Linux Networking (netdev01), Ottawa, Canada, Feb 2015
  9. Ryo Nakamura, Kouji Okada, Yuji Sekiya and Hiroshi Esaki, “ovstack : A Protocol Stack of Common Data Plane for Overlay Networks”, 1st IEEE/IFIP International Workshop on SDN Management and Orchestration (SDNMO 2014), Krakow, Poland, May 2014
- **International Conferences (poster presentation)**
  10. Ryo Nakamura, Yuji Sekiya and Hiroshi Esaki, “Implementation and Operation of User Defined Network on IaaS Clouds Using Layer3 Overlay”, 3rd International Conference on Cloud Computing over Service Science 2013 (CLOSER 2013) Poster session, Achen, Germany, May 2013
  11. Ryo Nakamura, Yukito Ueno, Katsuhiro Horiba, Yuji Sekiya, and Hiroshi Esaki, “Route Optimization for Geographically Distributed IaaS Platform Through the Integration of LISP and VXLAN”, AsiaFI 2012 Summer School,

Poster session, Kyoto, Japan, Aug 2012

12. Katsuhiro Horiba, Kazuya Okada, Yoshihiro Okamoto and Ryo Nakamura. "Adaptive Server Load Balancing on Distributed Cloud", In Asian Internet Engineering Conference (AINTEC) Poster session, November 2011

- **Domestic Conferences (w/ review)**

13. Yuta Tokusashi, Yohei Kuga, Ryo Nakamura, Hajime Tazaki, Hiroki Matsutani, "mitiKV: An Inline Mitigator for DDoS Flooding Attacks", インターネットコンファレンス, 東京, 2016 年 10 月
14. 中村 遼, 岡田 耕司, 重近 範行, 村井 純, "Universal P2P Architecture for feasible IP Multicast の設計と実装", インターネットコンファレンス, 東京, 2010 年 10 月

- **Domestic Conferences (w/o review)**

15. 中村 遼, 堀場 勝広, 関谷 勇司 "SDN を用いたクラウドサービスネット ワークの実現", 電子情報通信学会, 信学技報, vol. 113, no. 200, IA2013-17, pp. 5-10, 2013 年 9 月

- **Other Activities**

16. Interop Tokyo, ShowNet Network Operation Center (NOC) team member in charge of Layer-2, Layer-3 and Software Defined Networking, 2012 - 2016.

References

[1] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks.* Prentice Hall, 5 ed., 9 2010.

[2] "Internet2 noc support - combined infrastructure network map." http://noc.net.internet2.edu/i2network/maps-documentation/maps/internet2-combined-infrastructure-network-map.html. visited on 2016/9/21.

[3] B. Godfrey, "Jellyfish: Networking data centers randomly." http://youinfinitesnake.blogspot.jp/2012/04/jellyfish-networking-data-centers.html.

[4] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "Spain: Cots data-center ethernet for multipathing over arbitrary topologies," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, (Berkeley, CA, USA), pp. 18–18, USENIX Association, 2010.

[5] P. Baran, "On distributed communications networks," *IEEE Transactions on Communications Systems*, vol. 12, pp. 1–9, March 1964.

[6] P. Hoffman, "The tao of ietf: A novice's guide to the internet engineering task force." https://www.ietf.org/tao.html, 2012.

[7] J. Postel, "Internet Protocol." RFC 791, Mar. 2013.

[8] V. Fuller and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan." RFC 4632, Mar. 2013.

[9] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the aggregatability of router forwarding tables," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, March 2010.

[10] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis, "Smalta: Practical and near-optimal fib aggregation," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, (New York, NY, USA), pp. 29:1–29:12, ACM, 2011.

[11] R. J. Perlman, "Network layer protocols with byzantine robustness," 1988. Thesis (Ph. D.) – Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.

[12] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, pp. 277–288, Nov. 1984.

[13] Cisco Systems, "White paper: Cisco vni forecast and methodology, 2015-2020." http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html, January 2016.

[14] Nippon Telegraph and Telephone Corporation, "The state of fixed-line telephone services going forward." http://www.ntt.co.jp/news2015/1511ewbw/pdf/xddh151106d_all.pdf, November 2015.

[15] Y. Ajima, T. Inoue, S. Hiramoto, S. Ando, M. Maeda, T. Yoshikawa, K. Hosoe, and T. Shimizu, "The tofu interconnect 2," in *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*, pp. 57–62, Aug 2014.

[16] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray cascade: A scalable hpc system based on a dragonfly network," in *High Performance Computing, Networking, Storage and*

Analysis (SC), 2012 International Conference for, pp. 1–9, Nov 2012.

[17] Mellanox Technologies, "Top500 results and analysis." http://www.mellanox.com/page/top_500, 2015.

[18] S. O. Bradner and A. J. Mankin, "The Recommendation for the IP Next Generation Protocol." RFC 1752, Mar. 2013.

[19] "Measurements — world ipv6 launch." http://www.worldipv6launch.org/measurements/. visited on 2016/9/9.

[20] "Amsterdam internet exchange statistics." https://ams-ix.net/technical/statistics. visited on 2016/9/9.

[21] K. Lindqvist and J. Abley, "Operation of Anycast Services." RFC 4786, Oct. 2015.

[22] M. Lasserre, F. Balus, T. Morin, D. N. N. Bitar, and Y. Rekhter, "Framework for Data Center (DC) Network Virtualization." RFC 7365, Oct. 2015.

[23] B. Adamson, J. P. Macker, and R. Cole, "Definition of Managed Objects for the Mobile Ad Hoc Network (MANET) Simplified Multicast Framework Relay Set Process." RFC 7367, Oct. 2015.

[24] M. Mahalingam, T. Sridhar, M. Bursell, L. Kreeger, C. Wright, K. Duda, P. Agarwal, and D. Dutt, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks." RFC 7348, Oct. 2015.

[25] B. Davie and J. Gross, "A Stateless Transport Tunneling Protocol for Network Virtualization (STT)," Internet-Draft draft-davie-stt-08, Internet Engineering Task Force, June 2016.

[26] J. Gross, I. Ganga, and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation," Internet-Draft draft-ietf-nvo3-geneve-03, Internet Engineering Task Force, Sept. 2016.

[27] E. Rosen and R. Callon, "Multiprotocol Label Switching Architecture." RFC 3031, Mar. 2013.

[28] "Open compute project." http://www.opencompute.org/.

[29] R. Perlman, D. G. Dutt, A. Ghanwani, A. Banerjee, and D. E. E. 3rd, "Transparent Interconnection of Lots of Links (TRILL) Use of IS-IS." RFC 6326, Oct. 2015.

[30] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: A scalable ethernet architecture for large enterprises," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, (New York, NY, USA), pp. 3–14, ACM, 2008.

[31] 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," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, (New York, NY, USA), pp. 39–50, ACM, 2009.

[32] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, (New York, NY, USA), pp. 63–74, ACM, 2008.

[33] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, (Berkeley, CA, USA), pp. 17–17, USENIX Association, 2012.

[34] P. Lapukhov, "Building scalable data centers: Bgp is the better igp." https://www.nanog.org/meetings/nanog55/presentations/Monday/Lapukhov.pdf, June 2010. NANOG 55.

[35] A. Andreyev, "Introducing data center fabric, the next-generation facebook data center network." https://code.facebook.com/posts/360346274145943/introducing-data-

center-fabric-the-next-generation-facebook-data-center-network/, November 2014. Facebook, Inc.

[36] 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," *Commun. ACM*, vol. 54, pp. 95–104, Mar. 2011.

[37] J. Corbet, "Checksum offloads and protocol ossification." https://lwn.net/Articles/667059/.

[38] Intel, "Ethernet controller 10 gigabit and 40 gigabit xl710 family." http://www.intel.com/content/www/us/en/ethernet-products/converged-network-adapters/ethernet-xl710.html.

[39] Mellanox Technologies, "Mellanox connectx-3 pro product brief." http://www.mellanox.com/page/products_dyn?product_family=162.

[40] "Distance Vector Multicast Routing Protocol." RFC 1075, Mar. 2013.

[41] "IP in IP Tunneling." RFC 1853, Mar. 2013.

[42] D. Farinacci, S. P. Hanks, D. Meyer, and P. S. Traina, "Generic Routing Encapsulation (GRE)." RFC 2784, Mar. 2013.

[43] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "The Locator/ID Separation Protocol (LISP)." RFC 6830, Oct. 2015.

[44] P. Quinn and U. Elzur, "Network Service Header," Internet-Draft draft-ietf-sfc-nsh-10, Internet Engineering Task Force, Sept. 2016.

[45] 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, (New York, NY, USA), pp. 131–145, ACM, 2001.

[46] R. Nakamura, K. Okada, Y. Sekiya, and H. Esaki, "A common data plane for multiple overlay networks," *Comput. Netw.*, vol. 92, pp. 240–250, Dec. 2015.

[47] R. Nakamura, Y. Sekiya, and H. Esaki, "Implementation and operation of user defined network on iaas clouds using layer 3 overlay," in *Proceedings of the 3rd International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, pp. 366–369, 2013.

[48] "LISPmob: Mobile Networking through LISP." http://www.openoverlayrouter.org/lispmob/.

[49] "Linux containers." https://linuxcontainers.org/.

[50] M. Lasserre, F. Balus, T. Morin, D. N. N. Bitar, and Y. Rekhter, "Framework for Data Center (DC) Network Virtualization." RFC 7365, Oct. 2015.

[51] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture." RFC 7665, Nov. 2015.

[52] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, (Santa Clara, CA), pp. 523–535, USENIX Association, Mar. 2016.

[53] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture," in *Proceedings of the USENIX Winter 1993 Conference*, USENIX'93, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 1993.

[54] T. Herbert, L. Yong, and F. Templin, "Extensions for Generic UDP Encapsulation," Internet-Draft draft-herbert-gue-extensions-00, Internet Engineering Task Force, June 2016.

[55] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet-Draft draft-hamilton-quic-transport-protocol-00, Internet Engineering Task Force, July 2016.

[56] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it

still possible to extend tcp?," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, (New York, NY, USA), pp. 181–194, ACM, 2011.

[57] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, July 2014.

[58] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "Pisces: A programmable, protocol-independent software switch," in *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM '16, (New York, NY, USA), pp. 525–538, ACM, 2016.

[59] S. Boutros, A. Sajassi, S. Salam, D. Cai, S. Thoria, tsingh@juniper.net, J. Drake, and J. Tantsura, "VXLAN DCI Using EVPN," Internet-Draft draft-boutros-bess-vxlan-evpn-01, Internet Engineering Task Force, Sept. 2016. Work in Progress.

[60] J. Naous, G. Gibb, S. Bolouki, and N. McKeown, "Netfpga: Reusable router architecture for experimental research," in *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '08, (New York, NY, USA), pp. 1–7, ACM, 2008.

[61] J. Priko, "Hadware switches - the open-source approach," tech. rep., Netdev 0.1, The Technical Conference on Linux Networking, Ottawa, Canada, Feb 2015.

[62] J. H. Salim, "Linux Netlink as an IP Services Protocol." RFC 3549, Mar. 2013.

[63] Linux Foundation, "iproute2." http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2.

[64] L. Rizzo and M. Landi, "Netmap: Memory mapped access to network devices," in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (New York, NY, USA), pp. 422–423, ACM, 2011.

[65] Intel, "Intel data plane development kit." http://dpdk.org/.

[66] K. Yasukata, M. Honda, D. Santry, and L. Eggert, "Stackmap: Low-latency networking with the os stack and dedicated nics," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, (Denver, CO), USENIX Association, June 2016.

[67] "The fast data project." https://fd.io/.

[68] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe, "Arrakis: The operating system is the control plane," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, (Broomfield, CO), pp. 1–16, USENIX Association, Oct. 2014.

[69] G. Stark and S. Sezer, "Nfp-6xxx - a 22nm high-performance network flow processor for 200gb/s software defined networking," in *2013 IEEE Hot Chips 25 Symposium (HCS)*, pp. 1–21, Aug 2013.

[70] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy, "High performance packet processing with flexnic," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, (New York, NY, USA), pp. 67–81, ACM, 2016.

[71] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "Softnic: A software nic to augment hardware," Tech. Rep. UCB/EECS-2015-155, EECS Department, University of California, Berkeley, May 2015.

[72] J. T. Moy, "OSPF Version 2." RFC 2328, Mar. 2013.

[73] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, pp. 909–928, Second 2013.

[74] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers,"

in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, (New York, NY, USA), pp. 63–74, ACM, 2009.

[75] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, (New York, NY, USA), pp. 75–86, ACM, 2008.

[76] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: Topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, ACM, 2009.

[77] J. Kim, W. Dally, S. Scott, and D. Abts, "Cost-efficient dragonfly topology for large-scale systems," *IEEE Micro*, vol. 29, pp. 33–40, Jan 2009.

[78] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 68–73, Dec. 2008.

[79] A. Ford, C. Raiciu, M. J. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses." RFC 6824, Oct. 2015.

[80] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo, "Explicit path control in commodity data centers: Design and applications," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, (Oakland, CA), pp. 15–28, USENIX Association, May 2015.

[81] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 482–494, May 1998.

[82] Microsoft, "Windows Azure' s Flat Network Storage to Enable Higher Scalability Targets." http://blogs.msdn.com/b/hanuk/archive/2012/11/04/windows-azure-s-flat-network-storage-to-enable-higher-scalability-targets.aspx.

[83] "ns-3 project." http://www.nsnam.org/.

[84] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous, "Direct code execution: Revisiting library os architecture for reproducible network experiments," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, (New York, NY, USA), pp. 217–228, ACM, 2013.

[85] J. Yen., "Finding the k shortest loopless paths in a network," in *Management Science, 1971*, 1971.

[86] C. Raiciu, M. J. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols." RFC 6356, Oct. 2015.

[87] The IP Networking Lab (INL), "Multipath tcp - linux kernel implementation." https://www.multipath-tcp.org/.

[88] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (New York, NY, USA), pp. 266–277, ACM, 2011.

[89] J. Medved, S. Previdi, S. Ray, H. Gredler, and A. Farrel, "North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP." RFC 7752, Mar. 2016.

[90] "Quagga routing suite." http://www.nongnu.org/quagga/resources.html.

[91] M. Goyal, M. Soperi, E. Baccelli, G. Choudhury, A. Shaikh, H. Hosseini, and K. Trivedi, "Improving convergence speed and scalability in ospf: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 14, pp. 443–463, Second 2012.

[92] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new

algorithm and its implementation," *ACM Trans. Algorithms*, vol. 3, Nov. 2007.

[93] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "Mate: Mpls adaptive traffic engineering," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1300–1309 vol.3, 2001.

[94] S. Butenweg, "Two distributed reactive mpls traffic engineering mechanisms for throughput optimization in best effort mpls networks," in *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, pp. 379–384 vol.1, June 2003.

[95] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2010.

Acknowledgements

I would like to thank my adviser, Prof. Hiroshi Esaki for having supervised me for 5 years since my master course. Prof. Yuji Sekiya also had led me in both academic and practical areas. Their advice and valuable comments with huge background and skills depend on their experiences gave me good directions on my research life.

I am indebted to Prof. Shigeru Chiba. His accurate advice lets me realize a research way of thinking and the skills of technical writing. If I had not received his great guidance, I could not complete this thesis. I really appreciate his help. I also appreciate the invaluable comments on the thesis from Prof. Kenji Yamanishi, Prof. Tomohiro Kudo and Prof. Mari Inaba.

Hirochika Asai also gave me a lot of very incisive comments in various areas. His advice always encouraged me to progress my activities. I appreciate a great deal of his help. I also thank Kouji Okada and Katsuhiro Horiba. They had guided me on research, engineering, development, network operation and how to work with others when I started to study computer networking at Keio University, Murai laboratory.

I owe Interop Tokyo ShowNet NOC team members to conceive and execute this work. My research topics are derived from the very hard work at ShowNet. I really thank all NOC team members, in particular, layer-2, layer-3 and software-defined networking teams, Takayuki Watanabe, Tomoko Okuzawa, Shuichi Saito, Hiroyuki Tanahashi, Hiroki Kawaguchi, and Noriyuki Shigechika.

I would also like to express my gratitude to all the member of Esaki laboratory and my colleagues, especially, Hajime Tazaki, Tomohiro Ishihara, Kensuke Fukuda, Yohei Kuga, Masayoshi Mizutani, Kazuya Okada, Daisuke Miyamoto, Michio Honda, Takashi Tomine, Toshiaki Hatano, Yukito Ueno, Mizuki Kimoto, and Akira Kinoshita.

Finally, I would like to thank my family for supporting my research life.