Performance and Reliability in Open Router Platforms for Software-Defined Networking

VORAVIT TANYINGYONG

Licentiate Thesis in Information and Communication Technology
Stockholm, Sweden 2014

TRITA-ICT/ECS AVH 14:07
ISSN 1653-6363
ISRN KTH/ICT/ECS/AVH-14/07-SE
ISBN 978-91-7595-082-2

KTH School of Information and
Communication Technology
SE-164 40 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie licentiatexamen i informations- och kommunikationsteknik fredagen den 16 maj 2014 klockan 10.00 i Sal D i Forum, KTH Skolan för informations- och kommunikationsteknik, Isafjordsgatan 39, Kista.

Tryck: Universitetsservice US AB

Abstract

The unprecedented growth of the Internet has brought about such an enormous impact on our daily life that it is regarded as indispensable in modern era. At the same time, the underlying Internet architecture is still underpinned by principles designed several decades ago. Although IP networking has been proven very successful, it has been considered as the cause to network ossification creating barriers to entry for new network innovations. To support new demands and requirements of the current and the future Internet, solutions for new and improved Internet architectures should be sought.

Software-defined networking (SDN), a new modularized network architecture that separates the control plane from the data plane, has emerged as a promising candidate for the future Internet. SDN can be described as flow-based networking, which provides finer granularity while maintaining backward compatibility with traditional IP networking.

In this work, our goal is to investigate how to incorporate flow-based networking into open router platforms in an SDN context. We investigate performance and reliability aspects related to SDN data plane operation in software on open source PC-based routers.

Our research methodology is based on design, implementation, and experimental evaluation. The experimental platform consists of PC-based routers running open source software in combination with commodity-off-the-shelf (COTS) hardware components. When it comes to performance aspects, we demonstrate that by offloading the lookup from a CPU to a network interface card, the overall performance is improved significantly. For enhanced reliability, we investigate bidirectional forwarding detection (BFD) as a component to realize redundancy with fast failover. We demonstrate that BFD becomes unreliable under high traffic load and propose a solution to this problem by allocating dedicated system resources for BFD control messages. In line with this solution, we extend our architecture for next-generation PC-based routers with OpenFlow support by devising a strategy to efficiently map packet forwarding and application processing tasks onto the multi-core architecture on the PC-based router. This extension would make it possible to integrate BFD effectively into the router platform.

Our work demonstrates the potentials of open router platforms for SDN. Our prototypes offer not only high performance with good reliability but also flexibility to adopt new software extensions. Such platforms will play a vital role in advancing towards the future Internet.

Sammanfattning

Internets makalösa tillväxt har lett till en såpass stor inverkan på vårt dagliga liv att åtkomst till nätet idag betraktas som oumbärligt. Samtidigt är den underliggande arkitekturen bakom Internet fortfarande baserad på principer som utformades för flera decennier sedan. Även om IP-tekniken har visat sig vara mycket framgångsrik, anses den ibland även som en orsak till en viss stelhet som tenderar att skapa inträdesbarriärer för nya nätverksbaserade innovationer. För att stödja nya krav och behov i det nuvarande och framtida Internet bör man söka nya och förbättrade lösningar när det gäller Internets arkitektur.

SDN (Software-Defined Networking), en ny modulär nätverksarkitektur som separerar kontrollplanet från dataplanet, har kommit fram som en lovande kandidat för framtida Internet-lösningar. SDN kan beskrivas som en flödesbaserad nätverksteknik, vilken ger finare granularitet jämfört med traditionella IP-nätverk samtidigt som det är bakåtkompatibelt med existerande lösningar.

I det här arbetet är vårt mål att undersöka hur man kan införliva flödesbaserad nätverksteknik i öppna router-plattformar i ett SDN-sammanhang. Vi undersöker prestanda- och tillförlitlighetsaspekter relaterade till mjukvaruoperation av dataplanet i SDN på PC-baserade routrar.

Vår forskningsmetodik bygger på design, implementering, och experimentell utvärdering. Den experimentella plattformen består av PC-baserade routrar som kör programvara med öppen källkod i kombination med COTS-hårdvara (Commodity-Off-The-Shelf). När det gäller prestandaaspekter, visar vi att det prestandan som helhet förbättras avsevärt genom att avlasta paketuppslagning från en processor till hårdvara på ett nätverkskort. För ökad tillförlitlighet undersöker vi BFD (Bidirectional Forwarding Detection) som en komponent för att implementera redundans med snabb omkoppling vid fel. Vi visar att BFD blir otillförlitlig under hög trafikbelastning och föreslår en lösning på detta problem genom att avsätta särskilda systemresurser för kontrollmeddelanden i BFD. I linje med denna lösning, utökar vi vår arkitektur för nästa generations PC-baserade routrar med stöd för OpenFlow genom att utarbeta en strategi för att effektivt kombinera pakethantering och tillämpningsprogramvara på en flerkärniga processorarkitektur i PC-baserade routrar. Denna lösning gör det möjligt att effektivt integrera stöd för BFD i vår routerplattform.

Vårt arbete visar möjligheterna med öppna router-plattformar för SDN. Våra prototyper erbjuder inte bara hög prestanda med god tillförlitlighet, utan också flexibilitet när det gäller att stödja nya mjukvarutillägg. Sådana plattformar kommer att spela en vital roll i vägen till ett framtida Internet.

Acknowledgements

This thesis could not have been materialized without the help and support of my supervisors, Peter Sjödin and Markus Hidell, who spent substantial efforts in giving guidance and overseeing all of my research. No gratitude would suffice. I would also like to thanks my newly appointed supervisor, Jeong-woo Cho, who gave valuable feedback on this thesis and partook in constructive discussions on what to come ahead.

A special thank goes to Björn Pehrson, who trusted in my competence and has brought me into the academic environment at KTH. I am also grateful to Robert Olsson, who always shares his expertise and gives valuable insights.

Many thanks go to everyone at NSLab, whom I have shared my academic journey through thick and thin with. Without a doubt, "*a friend in need is a friend in deed*"

I would also like to thank all of my colleagues and fellow researchers, whom I have worked and/or shared wonderful discussions with.

Finally, I sincerely thank my friends and family who always lend me their ears and shoulders when I need them the most.

Contents

List of Abbreviations

API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ASSP	Application-Specific Standard Product
BFD	Bidirectional Forwarding Detection
COTS	Commodity-Off-The-Shelf
CPU	Central Processing Unit
DC-DC	Direct Current to Direct Current
DCN	Data Center Network
DHT	Distributed Hash Table
FPGA	Field-Programmable Gate Array
GPP	General Purpose Processor
GPU	Graphics Processing Unit
IBT	Input Batching Threshold
IP	Internet Protocol
ISP	Internet Service Provider
MPLS	Multiprotocol Label Switching
NFE	Network Flow Processor
NIC	Network Interface Card
NPU	Network Processing Unit
OAM	Operations, Administration, and Maintenance

OS	Operating System
PC	Personal Computer
PLD	Programmable Logic Device
RFC	Recursive Flow Classification (Algorithm)
RFC	Request for Comments (Standard)
SDN	Software-Defined Networking
SRAM	Static Random-Access Memory
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VPN	Virtual Private Network

Part I

Thesis Overview

Chapter 1

Introduction

This chapter describes our research motivation for this licentiate thesis work. It also provides a brief summary of the main contributions of this thesis and an outline of how the remaining chapters of this thesis are organized.

1.1 Research Motivation

The unprecedented growth of the Internet has brought about such an enormous impact on our daily life that it is regarded as indispensable in modern era. Technology advancements have led to an explosion of mobile devices and today a large amount of users possess multiple end terminals with multiple broadband access technologies running concurrently [16, 11]. This poses high demands for high network capacity and at the same time increases the complexity of the underlying network technologies. As the demand for Internet access grows, the underlying infrastructure needs to be able to keep up with the ever-increasing amounts of data traffic. This calls for revising the current Internet architecture.

Despite the tremendous growth of the Internet, the underlying Internet architecture is still underpinned by the original principles designed decades ago [19]. The Internet Protocol suite (also known as TCP/IP) is used as communication protocols for the Internet having IP as the key network protocol. Traditionally, the Internet uses a best-effort IP destination-based forwarding scheme. Although IP networking has been proven very successful, it has been considered as the cause to network ossification creating barriers to entry for new network innovations [1, 67]. In addition, IP suffers from many shortcomings when it comes to security, mobility, and service diversification [19, 67]. Although there are attempts to address these issues, they are relatively complex and not widely deployed [21, 46, 10].

To meet the increasing user demand, network operators and internet service providers (ISP) use server virtualizations and cloud services to improve overall utilizations of their equipment. Although these technologies offer competitive advantages and better services to the customers, they add complexity to the underlying

network. They also lead to even higher requirements in various aspects such as security, scalability, and service availability.

Furthermore, an ISP network is likely to be heterogeneous containing various devices with different implementations from various vendors. This poses a challenge for control and management. For instance, each device may have its own interpretation of the same control message and react to it differently. In order to maintain a consistent policy in a network, the network operator is required to understand how each individual device functions in order to configure each device accordingly. This makes it difficult to scale the network. As the network grows and more devices are added, it is more prone to inconsistent policies and unintentional errors.

To support new demands and requirements of the current and the future Internet, a new and improved Internet architecture is needed. The Internet architecture requires a mechanism to facilitate new network innovations to overcome network ossification. It also needs to be able to adapt to new demands, changes in traffic patterns, as well as support new networking technologies, protocols, and tools. Moreover, it is crucial for the network control function to be vendor-independent. Such a property enables dynamic control and management of network devices and provides support for an automated mechanism to ensure network-wide consistent policies at scale. In this context, the concept of software-defined networking (SDN) [43] emerges as a promising candidate.

SDN is a new modularized network architecture that separates the control plane from the data plane. By doing so, it creates abstraction layers for the control plane and the data plane. Each layer can evolve independently, which enables a faster development process of network devices and at the same time allows new services and innovations to flourish. The interaction between the layers is done through a communication protocol that is an open vendor-independent API. The interaction between the layers, in theory, can be one of any standard communication interfaces between the control plane and the data plane. However, in reality, OpenFlow [42] is the first and most dominant standard communication interface for SDN.

SDN uses flow-based networking, a forwarding scheme that makes forwarding decision based on flows. A flow is a generic way to describe the traffic that offers finer granularity while maintaining backward compatibility with traditional IP networking. These properties enable flow-based networking to be a powerful forwarding scheme that offers flexibility and facilitates new forwarding services.

Our research goal is to investigate how to incorporate flow-based networking into open router platforms in an SDN context. Within the scope of this work, we investigate a set of challenges related to SDN data plane operation in software on open source PC-based routers including performance and reliability aspects.

1.2 Thesis Contribution

We demonstrate that a next-generation PC-based router with support for flow-based networking in an SDN context can be realized using open source software

on commodity-off-the-shelf (COTS) hardware components. We design, implement, and evaluate a forwarding architecture for next-generation PC-based routers that uses OpenFlow switching as a forwarding engine. To improve OpenFlow lookup performance, we introduce a fast data path based on caching of flow table entries in the on-board classification hardware on the network interface card (NIC). This design is simple to implement yet offering significant throughput improvements when compared to regular software-based OpenFlow switching.

For enhanced reliability, we investigate bidirectional forwarding detection (BFD) as a component for redundancy with fast failover. We demonstrate that the software implementation of BFD becomes unreliable under high traffic load and propose a solution to this problem by allocating dedicated system resources for BFD control messages. In addition, we devise a strategy to utilize multi-core architecture on the PC-based router that allows us to extend our architecture to perform other tasks beyond solely routing. In this context, our suggested extension would make it possible to integrate BFD effectively into the architecture.

1.3 Thesis Organization

This licentiate thesis is a compilation thesis. It comprises two parts. The first part provides an overview of the thesis and a description of the problem statements. The second part is a compilation of the published, peer-reviewed papers of the author of this thesis.

The rest of the first part is organized as follows. Chapter 2 provides the necessary background to understand this thesis. Chapter 3 defines the problem addressed in this thesis. Chapter 4 presents a list of publications of the author and summarizes the contributions of this thesis. Finally, Chapter 5 concludes the thesis and provides directions for future work.

Chapter 2

Background and Literature Review

This chapter aims to provide the necessary background to understand the rest of the thesis. The descriptions given in this chapter present a generic view of the concepts and are focused on the most relevant details related to this thesis work, which includes flow-based networking, OpenFlow switching technology, and software-defined networking (SDN).

2.1 Flow-Based Networking

While traditional IP networking uses a forwarding scheme based on destination IP addresses, flow-based networking is a networking scheme that makes forwarding decision based on flows. A flow is a generic way to describe the traffic. For instance, a flow can be described as all traffic to KTH, all traffic between Alice and Bob, all UDP traffic, or all IP traffic to a specific destination. This type of generic description is very intuitive and can be expressed using multiple fields in the packet headers. Moreover, it can also describe the traffic in the same way as a traditional IP network does with the longest destination prefix match since IP destination address field is a subset of the packet header fields.

Using flows to characterize the traffic offers finer granularity compared to longest prefix match while maintaining backward compatibility to traditional IP networking. This means that flow-based networking is transparent to the end hosts and it allows existing protocols to function normally without any modification. In addition, it is compatible with existing network virtualization techniques such as virtual local area networks (VLANs) [25] and virtual private networks (VPNs) [33]. These properties enable flow-based networking to be a powerful networking scheme that offers flexibility and fosters new forwarding services.

In order to realize flow-based networking, the traditional IP forwarding engine on a router must be replaced with a new forwarding engine that can make forwarding decisions based on multiple fields in the packet header. In this context, OpenFlow switching technology [32] is considered as a key component to enable

flow-based networking.

2.2 OpenFlow Switching Technology

OpenFlow [42] is a novel technology that started out from an initiative to provide a way for researchers to run experimental protocols concurrently with production networks [32, 55]. OpenFlow achieves this by enabling flow tables in switches and routers to be programmed via a standardized interface, namely the OpenFlow protocol. This protocol allows separation of the control plane and the forwarding plane. A flow table, which is available in most switches and routers, is programmed to partition the traffic into flows based on predefined rules. This allows researchers to be able to run experiments without disturbing the production networks. As a result, it opens up possibilities for new ideas to flourish.

Switches and Routers that support the OpenFlow protocol are called *OpenFlow switches*. OpenFlow switches are classified into two types: *OpenFlow-only* switches, which support only OpenFlow operation and *OpenFlow-hybrid* switches, which support both OpenFlow operation and normal Ethernet switching operation.

2.2.1 Main Components

According to the latest OpenFlow specification [44], an OpenFlow switch is comprised of three main components, as illustrated in Figure 2.1, which are 1) one or more flow tables and a group table that perform packet lookups and forwarding, 2) an OpenFlow channel that is a communication interface connecting the switch to an external controller, and 3) an OpenFlow protocol that provides a standard way for a switch to communication with the controller as well as for the controller to manage the switch.

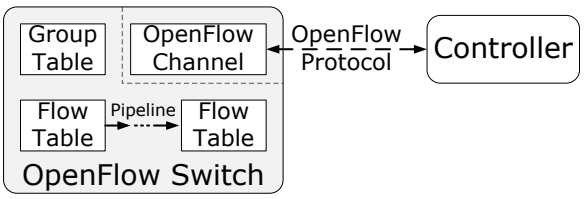

Figure 2.1: OpenFlow Switch Technology. (Adapted from [44] Figure 1)

2.2.2 Packet Lookup

The packet lookups in OpenFlow start from the first flow table and may continue to more flow tables as per instructions defined through the OpenFlow pipeline processing. Each flow table contains multiple flow entries. Each flow entry has *match fields* used in the packet lookup matching and a set of *instructions* to apply to the

matching packet. The matching is done in priority order, where the first matching entry is used. If a match is found, the associated instructions are executed. Otherwise, the packet lookup follows instructions in the table-miss flow entry, which, for example, could be to drop the packet, forward it to the controller or pass the packet to a subsequent table. A flow entry can also point to a group entry in the group table to provide additional methods of forwarding such as for multicast, broadcast, and fast failover.

In OpenFlow, match fields are defined based on information from multiple protocol layers in multiple fields of the packet header as illustrated in Figure 2.2. They also include an ingress port and a metadata value, which is used to carry information across tables. Wildcards can be used for any of these fields, which makes OpenFlow very flexible for flow classification and lookup. A default set of match fields can be found in [44].

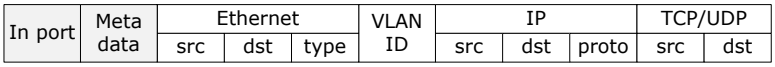

Figure 2.2: Match fields in Packet Header

2.2.3 Limitations

To adopt OpenFlow technology into an open source software router, a software implementation of an OpenFlow switch is needed. There are many variants of OpenFlow software implementations both in kernel space and user space. Kernel space implementations offer higher performance but are more difficult to setup when compared to user space implementations. Examples of OpenFlow implementations are the original Stanford software reference [63], which is now only available as a user space implementation, Open vSwitch [45], which is a multilayer virtual switch that offers both kernel space and user space implementations.

Although OpenFlow lookup is general and flexible, it inherits lookup performance penalties that come with software processing. OpenFlow switching implies that more information needs to be taken into account in the packet lookup compared to regular IP forwarding. With multiple fields, the packet lookup becomes a difficult challenge [23, 62]. In addition, OpenFlow's switching flexibility, gained by allowing wildcards on any field in the lookups, results in reduced performance when implemented in software since such implementations often come down to doing exhaustive searches. In fact, such effects on the performance can be seen in the OpenFlow switching performance experiments done in [3, 59].

2.2.4 Related Work on Improving OpenFlow Lookup Performance

The implementation of OpenFlow lookups can have a large impact on the overall performance of the OpenFlow switch, especially when the system is under high traffic load. Depending on the existing flow entries on the OpenFlow switch, the OpenFlow lookup can be done solely in the data plane operation when a match is found. If no match is found, the packet header may be passed on to the controller, which, in turn, performs the lookup and decides what to do with the packet. Regardless of the situation, an additional mechanism to assist the lookup process is needed to ensure high performance.

Several research studies propose to enhance software-based OpenFlow lookups by using specialized hardware to accelerate the performance. Naous et al. [34] implement an OpenFlow switch on the NetFPGA platform, which is an open source programmable platform designed for research and classroom experimentation [37]. Their implementation is capable of running at line-rate across the four 1GB NetFPGA ports. It can accommodate more than 32,000 exact match flow entries with a possible expansion to double this number. Research studies in [31, 39, 20] suggest to offload OpenFlow lookup using solutions based on network processors. Luo et al. [31] uses the Netronome NFE-i8000 network processor acceleration card. Their implementation reduces packet delay by 20%. However, the packet forwarding throughput is still comparable to the conventional OpenFlow kernel module implementation due to the slow speed of the network processor on their implementation. On the other hand, Neugebauer [39] has another implementation based on Netronome NFE cards with NFP-32xx (Network Flow Processor) [38]. The prototype implementation based on Open vSwitch offers at least 8-10 times performance improvement over software-only Open vSwitch. Ferkouss et al. [20] implement OpenFlow version 1.1 on the EZchip NP4 [17] network processor. To enhance classification performance, they use static random-access memory (SRAM) and Ternary content-addressable memory (TCAM) side by side to pipeline the lookup. This is done by mapping multiple flow tables to different types of memories. Several designs of chaining the lookups are described with a notable design based on the recursive flow classification (RFC) being the best in term of flexibility.

Regarding hardware processing units for accelerating performance, Sezer et al. outlines the trade-off between programmability/flexibility and performance in [51]. The overall conclusion is that the more specialized the hardware is the higher performance it can offer at a cost of lower programmability. In their work, they consider different types of hardware processing units ranging from general purpose hardware to application-specific hardware, which includes general-purpose processors (CPUs/GPPs), network flow processors (NPUs/NFPs), programmable logic devices (PLDs) or field programmable gate arrays (FPGAs), application-specific standard products (ASSPs), and application-specific integrated circuits (ASICs).

Nevertheless, modern commodity hardware components are still very capable despite being built for general-purpose. Several research studies have demonstrated

that they can be used to increase the overall packet processing performance. These solutions are generic and can be adopted for offloading the OpenFlow lookups. For instance, PacketShader [24] uses Graphics Processing Unit (GPU) acceleration to offload computation and memory-intensive workload. ServerSwitch [30] uses a commodity switching chip to build a customized NIC that can perform various customized packet forwarding in recently proposed data center network (DCN) designs, and leverages server CPU for control and data plane packet processing. FIBIUM [49] proposes a hardware accelerated software router that uses commodity PC hardware for control path and couples it with programmable switching hardware for data path. Flowstream [22] leverages packet forwarding to the switching hardware by consolidating commodity PC hardware and programmable commodity switching hardware. RouteBricks [18] builds a scalable software router using clusters of general-purpose PC hardware that parallelizes router functionality across multiple servers and across multiple CPU cores within a single server.

In addition, a software implementation such as netmap [48] can also be used to improve the lookup performance. Netmap [48] is a novel framework that enables applications to handle high packet throughput without requiring custom hardware or changes to applications. The performance gains are made through eliminating the main processing costs by pre-allocating resources, batching tasks, and sharing buffers and metadata between kernel and user space.

2.3 Software-Defined Networking

Software-defined networking (SDN) [43] is a new modularized network architecture that separates the control plane from the data plane. Unlike a traditional IP network architecture that normally has a distributed control logic residing on network devices, the control logic in SDN is moved to a centralized unit, often called a *controller*, while the data plane remains on the physical network devices as shown in Figure 2.3.

This decoupling creates distinct abstraction layers and simplifies the tasks in each layer. Each layer can evolve independently, which enables a faster development process of network devices. Furthermore, an additional abstraction layer can be introduced at the control layer to separate network services from the high-level business oriented services, which simplifies network operation and management. The interaction between the layers is done through a communication protocol that is an open vendor-independent API. The SDN architecture is illustrated in Figure 2.4.

The *Infrastructure Layer* contains the physical network infrastructure. The data plane resides in this layer and is simply the packet-forwarding path through the network devices. Removing the control plane simplifies network devices, since they no longer need to support an extensive list of protocols as they traditionally have to. In an SDN context, the network device is called a *switch*. Thus, this terminology will be used to refer to a network device in an SDN context.

The *Control Layer* contains the logically centralized software-based SDN con-

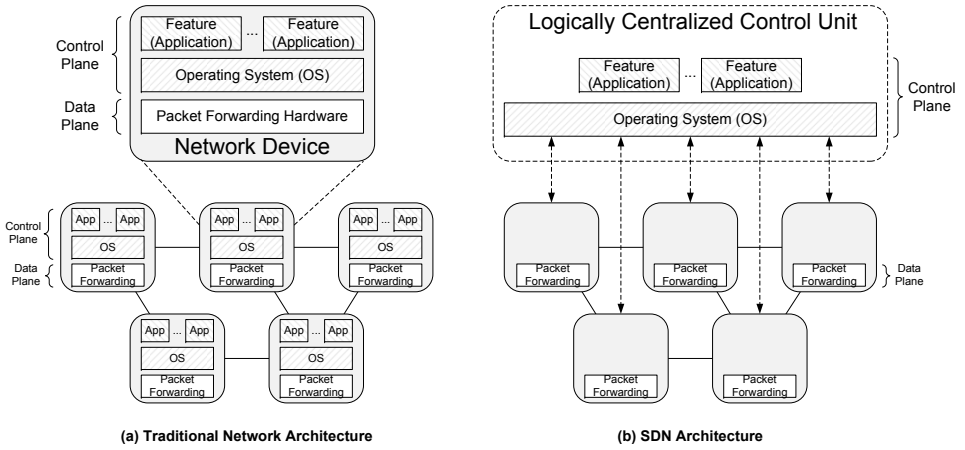

Figure 2.3: Control Plane and Data Plane on Network Devices

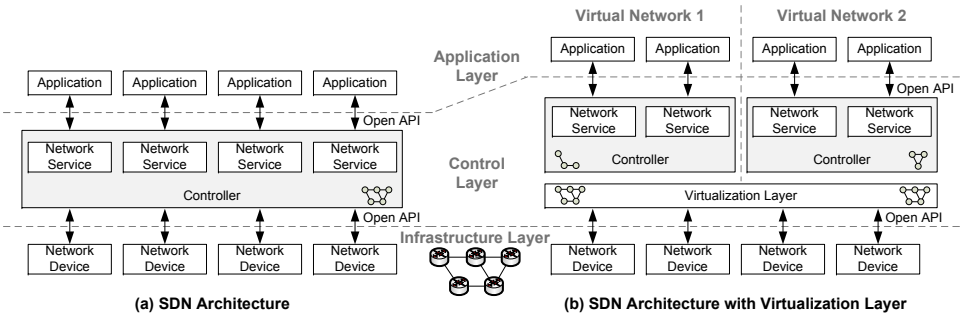

Figure 2.4: Software-Defined Networking Architecture

troller, which governs the control plane of the network. The controller is sometimes referred to as a *network OS*. It has a global view of the network state and is hosting network services such as topology discovery, routing, and spanning tree. Having the controller as a single logical unit simplifies the network control and management. It permits the network to be programmatically configured, ensures consistent policies across the network, and enables high flexibility in the control and management. The controller can communicate with the data plane through an open vendor-independent API such as OpenFlow [32]. For instance, given the global view of the network, the controller can compute forwarding decisions without having to deal with the distributed states of the switches in the underlying infrastructure. After computation, the controller can programmatically propagate the control informa-

tion in a form of forwarding instructions to the switches through the open API faster and less error-prone than manually configured by human.

The most well-known controller in the research community is NOX [40], which is the first (and one of the most commonly used) controller implemented as a user space open source software. Nevertheless, there are many implementations of the controller from both commercial vendors (such as Big Network Controller [4] from Big Switch Networks [5] and ProgrammableFlow Controller [36] from NEC [35]) and open source community (such as POX [41], Trema [66], Beacon [14], and Floodlight [47]).

In addition, an abstraction layer can be added at the control layer to create a virtualization layer. Network resources as well as the logically centralized control unit can be sliced and shared transparently and independently allowing users to use the network concurrently without affecting one another. Exemplary showcases of this are described in [56], in which FlowVisor [55] is introduced as a transparent layer that permits the physical network to be sliced enabling the production network to run alongside experiments without causing disruption in various campus networks.

The *Application Layer* contains higher-level business-oriented applications. At this layer, the applications running on top of the controller will see the network as a single logical switch. This abstraction enables applications to be independent from the detailed implementation in the controller, thereby making it easier to leverage network services and capabilities. For instance, access control, monitoring, and other policies can be applied to the single logical switch abstraction of the network. However, the API between the control and application layers is, in practice, still coupled to the controller and is not interoperable across different controllers.

Open API for control plane/data plane communications, in theory, can be one of any standard communication interfaces between the control plane and the data plane of SDN. In reality, OpenFlow is the first (and most dominant) standard communication interface defined thus far [43].

2.3.1 SDN operation

In SDN, the switches are controlled by the centralized controller. In order to forward a packet, the switch requires a forwarding rule (a flow entry) that matches the incoming packet. The controller is responsible for installing, updating, and removing the flow entries on the switches. The process of SDN operation is illustrated in Figure 2.5. Once a packet arrives to the switch, it checks for a match in its flow tables. If a match is found, the packet is processed according to the associated instructions defined in the matched flow entry. If the incoming packet is a new flow and no match is found then the switch may generate a request to the controller to decide on what to do with the packet. The controller makes a decision and responds with a new forwarding rule. Note that the controller can be proactive and install the rule on the downstream switches along the path to the destination. Finally, the switch installs the new rule and forwards the packet accordingly.

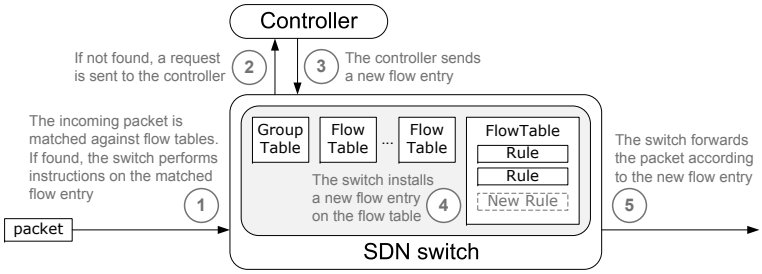

Figure 2.5: SDN Operation

2.3.2 Challenges from Decoupling Control Plane from Data Plane

Although decoupling the control plane from the data plane creates distinct abstraction layers and simplifies network operation and management, it leads to challenges on other aspects. In this section, only selected challenges relevant to this thesis work are described. Security aspect is not covered in this thesis. A survey on SDN security can be found in [50]. More elaborated descriptions on the key challenges of SDN can be found in [51, 9, 8].

Scalability

Moving the control plane to a centralized controller raises a major concern for the scalability of SDN. Since the controller resides on a remote host in the network, it can potentially become a bottleneck. Moreover, the probability increases as the size of the network grows since it means that the controller will have to handle more requests and more switches. This is especially critical in a network with many unique active flows since each new flow, which does not match any entry on the switch, requires to be initialized by the controller.

Latency

Moving the control plane to the centralized controller adds additional delays in the interaction between the control plane and the data plane, which can cause a negative effect on the overall network's reactions to changes and disruptions. Moreover, SDN requires all switches to exchange information with the controller, which could add more delay to the overall communication. High latency could cause the network to become less robust and reduce its reliability.

Interoperability

SDN requires interoperability at various levels. First, a standardized common API between the control plane and data plane is needed. Although SDN architecture

defines that a standard open API should be used for the interaction between the control plane and data plane, it is not trivial to achieve since this means that all switch vendors must adopt the same standard API. Otherwise, the network might still be bound to a vendor-specific implementation. In addition, the API needs to support various architectures that may have different requirements as well as to facilitate the independent evolution of both control plane and data plane in order to keep up with changes and foster innovations.

Next, SDN needs to be able to operate in harmony with the existing network infrastructure that is not SDN-enabled. A network is likely to have a large number of traditional IP network devices. Thus, it is unlikely that the network is rebuilt completely based on SDN technology; rather it is more realistic to have a gradual transition from a traditional IP network to SDN. Such a transition requires support of protocols and services in both traditional IP networks and SDN.

Last, the controllers should be interoperable. There are many implementations of SDN controllers, however they tend to consider themselves as the sole controller of SDN and do not have a standard API for the communication among different implementations of the controllers.

2.3.3 Related Work on Improving Scalability of SDN

In this section, we focus on the improvement of the control plane (i.e., the controller) in SDN. The improvement of the data plane (i.e., the switch) is directly related to the OpenFlow lookup. Thus, we refer to Chapter 2.2.4 for related work on improving OpenFlow lookup performance.

Yeganeh et al. gives an overview of the scalability concerns in SDN [68]. They argue that these concerns are not unique to SDN and could be addressed without losing the benefits of SDN. Moreover, they point out opportunities and challenges in scalability beyond traditional performance metrics, which includes orthogonal aspects such as manageability and functional scalability.

Various approaches have been taken to address the scalability issue of the controller. One approach is to improve the overall performance of the controller by utilizing performance optimization techniques and enhancing the architectural designs. For example, NOX-MT [65] utilizes multiple threads and optimization techniques such as I/O batching to improve the performance that outperforms NOX by a factor of 33 on a server with two quad-core 2GHz processor. Beacon [14], an OpenFlow controller implemented in Java, achieves high packet (flow request) processing performance by utilizing multi-threaded processing, run-to-completion reading design, and message batching writing design. The evaluation in [15] shows that the Beacon can scale linearly with processing cores and can handle 12.8 million packets per second with 12 threads. Maestro [7] tries to optimize the controller performance while maintaining the balance between fairness, latency, and throughput. It uses a workload-adaptive request batching algorithm called Input Batching Threshold (IBT), which uses measured throughput and latency at runtime to control the dynamic adaptation to achieve low latency at high throughput. Multiple

workload distribution designs for Maestro are evaluated in [6]. The results show that the round-robin workload distribution achieves near optimal fairness while maintaining good scalability.

Shah et al. [52] study the architectures of four prominent open source SDN controllers: NOX [40], Beacon [14], Maestro [7], and Floodlight [47]. They compare these controllers under different metrics and identify architectural guidelines for improving scalability of existing controllers or for designing new ones. The key guidelines are 1) for high throughput, use static switch partitioning and packet batching, 2) for delay-sensitive, use workload adaptive packet batching and task batching to reduce per-packet latencies, and 3) for further improvement on latency, the controller should send each outgoing control message individually.

Shalimov et al. presents in [53] a comprehensive analysis of popular open source SDN controllers in term of performance, scalability, reliability, and security. They found that the current controllers have bad scaling over the CPU cores. Moreover, they found that the majority of the controllers do not perform the expected behavior upon receiving a malformed message. They conclude that modern open source SDN controllers are still not ready for use in production. Thus, there are still rooms for improvement.

Another approach to improve the scalability of the controller is to make the controller distributed. Tootoonchian and Ganjali propose HyperFlow [64], a distributed event-based controller that allows multiple controllers in the network. By passively synchronizing network-wide views, all controllers share the same consistent view of the network. It localizes all decisions to each controller to minimize control plane response time. Koponen et al. propose Onix [28], a distributed control platform that handles state distribution and provides a general API for network control. To improve scalability, Onix utilizes distributed hash tables (DHT) to partition, and aggregate the network. It uses inconsistency detection and conflict resolution logic to provide consistency and a replicated database to provide durability. Dixit et al. propose ElastiCon [13], an elastic distributed controller architecture, in which a pool of controllers can dynamically grow or shrink according to traffic conditions. They also propose a novel switch migration protocol for dynamically shifting the load across the controller nodes.

Others take a different approach by keeping the packet processing task in the data plane to reduce the involvement of the controller. DIFANE [69] offers a scalable and efficient solution by partitioning the (flow matching) rule space into multiple subsets and dividing them across the switches. DIFANE selectively redirects packets through intermediate switches that store the necessary rules to keep packet processing in the data plane and reduces the burden on the centralized controller. Taking another approach, DevoFlow [12] reduces the communication between the switch and the controller by allowing a set of local actions that a switch can use without invoking the controller for flow-setup. In addition, it clones the wildcard rule to create a new microflow-specific rule, which is kept in an exact-match lookup table to reduce the use of the TCAM.

In line with the scalability issue, other research studies focus on resiliency and

fault management of SDN. In [2], Beheshti and Zhang propose to improve the resiliency of SDN by pre-configuring backup links in the switches. If a failure occurs, the switch makes a local action to reroute the control traffic through an alternate path to the controller. In addition, they propose algorithms for resilience-aware controller placement and control-traffic routing to maximize the possibility of fast failover. Kempf et al. propose a scalable fault management for OpenFlow in [27]. By keeping parts of the control operations colocated with the forwarding hardware, they incorporate onto the switch the connectivity monitoring function of the operations, administration, and maintenance (OAM) in the MPLS transport network. In addition, they introduce a general message generator and processing function on the switch to keep the fault management in the data plane. They demonstrate that such an implementation ensures fast recovery in a scalable way. Kuźniar et al. propose AFRO (Automatic Failure Recovery for OpenFlow) [29], a system that provides automatic failure recovery on behalf of simpler, failure-agnostic controller modules. AFRO does this by recording the events observed by the controller during normal operation. When failure occurs, AFRO creates a new isolated instance of the controller (called *Shadow controller*) on a *Shadow network*, an emulated network that is a copy of the actual network without the failed elements. Then, AFRO replays the recorded events on the Shadow controller to create new forwarding state. After that, AFRO computes a minimal set of rule changes and updates all switches. Finally, the controller state is replaced with the state in the Shadow controller. In [54], Sharma et al. study two mechanisms for fast failure recovery for OpenFlow, namely, restoration and protection. With restoration, an alternative path is established by the controller when it receives a failure notification from the switch. With protection, on the other hand, two disjoint paths (working path and protected path) are established in advance. When the failure is detected in the working path, the traffic is redirected to the protected path. The bidirectional forwarding detection (BFD) protocol is used to trigger the path switching in the protection mechanism. The results from their experimental evaluation show that a fast recovery time of less than 50 ms can be achieved with the protection mechanism.

Chapter 3

Problem Definition

Our research goal is to investigate how to incorporate flow-based networking into an open source software router in an SDN context. Within the scope of this work, we set out to solve challenging problems related to SDN data plane operation in software on open source PC-based routers. Specifically, we investigate performance aspect and reliability aspect.

Performance

Packet forwarding in SDN depends on the OpenFlow lookup, which is a complex and demanding task especially when the lookup is implemented in software. Given that there can be many unique active flows in the network, the OpenFlow lookup could take substantial amount of time when using software-based searching for a matching flow entry. Thus, it is crucial to investigate how to ensure that the packet forwarding in SDN can maintain good performance under high traffic load.

Reliability

In SDN, where the control plane is separate from the data plane, there is apparently an additional communication delay in the interaction between the two planes. Such a delay is undesirable in a network since it increases the overall response time. The issue could be aggravated further when a failure occurs. Traffic loop and/or black hole could happen before the controller is aware of the failure. Thus, it is crucial to investigate how to enable fast failure detection and ultimately to ensure a fast failover in SDN.

Moreover, we elaborate on the use of open source in combination with COTS in general. In today's networks, PC-based routers are often used as a competitive alternative to commercial hardware routers. However, traditional PC-based routers are usually optimized for IP networking with the longest destination prefix match and might not be optimal for SDN since SDN is fundamentally different in the

design and in its packet processing operations. This calls for further studies on how best to use standard PC hardware as an underlying SDN infrastructure.

Approaches to the Problems

Performance

The main objective in this part of the research is to investigate OpenFlow lookup performance and, if possible, propose improvements. In this work, we focus on software implementations of OpenFlow. We investigate and propose a method to integrate SDN by adding OpenFlow support to a PC-based router architecture to provide support for flow-based networking [57].

Although the software-based OpenFlow reference implementation is general and flexible, it inherits lookup performance penalties that come with software processing. OpenFlow switching implies that more information has to be taken into account in the packet lookup compared to regular IP forwarding. In addition, OpenFlow's switching flexibility, underpinned by wildcard lookups, results in reduced performance when implemented in software since such implementations often come down to doing exhaustive searches. In fact, such effects on the performance can be seen in the OpenFlow switching performance experiments done in [3]. To enhance software-based OpenFlow lookups, an additional mechanism to accelerate the performance is needed.

In this work, OpenFlow switching technology is integrated into PC-based router architecture to create next-generation routers that are more flexible than traditional IP-based routers. We aim at keeping our design open and accessible by using open source software and standard PC hardware components. From the software side, the publicly available software-based OpenFlow implementations [63, 45] match well with our purpose. From the hardware side, COTS hardware components are used to construct a PC-based router. Advancements in today's technologies enable COTS hardware components to provide performance enhancements that can be compared with specialized hardware. Moreover, they offer several advantages in term of price, flexibility, and programmability. Thus, we investigate COTS hardware components with on-board classification and propose an extension mechanism to accelerate OpenFlow's software lookup [58, 59].

Reliability

The main objective in this part of the research is to investigate bidirectional forwarding detection (BFD) as a component to realize redundancy with fast failover. The ultimate goal is to integrate BFD into network devices supporting SDN.

In SDN, distinct abstraction layers are created from decoupling the control plane from the data plane. Although this simplifies the network design and operation, it adds additional delays in the interaction between the control plane and the data plane, which might cause a negative effect on the overall network's reactions to

changes and disruptions. A mechanism to accelerate this process in SDN is undoubtedly needed.

BFD is a low-overhead, routing protocol independent protocol to detect failure in a path between adjacent forwarding engines. According to RFC 5880 [26], BFD is intended to be implemented as a part of the forwarding engine of a system should the forwarding and control engines be separated. This means that BFD, by design, is compatible with SDN, in which the control plane and the data plane are separated. Thus, we study the BFD protocol and investigate the trade-offs between short reaction times for rerouting and low probability of false alarms [61]. This is challenging to achieve under high traffic loads since the load can affect the BFD session which is formed to monitor the bidirectional forwarding capabilities.

Another challenge is how to integrate BFD into PC-based routers with flow-based networking support including a software implementation of OpenFlow. In such a platform, BFD also have to be implemented as a software process. Contentions for resources are bound to happen in this situation since system resources are shared among all the processing tasks. The overall performance depends on how resources are allocated and configured on the PC-based routers. In this context, we aim to utilize parallel processing capabilities that come with modern PC architectures (including multiple multi-core processors, multi-queue NICs, and dedicated memory nodes) to extend services offered beyond solely traditional routing and forwarding. We take into account a scenario of a combined server/router, which can act both as application server and packet forwarding unit [60].

Chapter 4

Thesis Contribution

This chapter summarizes the contributions of this thesis work. First, the publications of the author are listed. Then, the contributions are briefly described.

4.1 Publications

List of publications included as part of this thesis:

- **Paper A:** G. Su, M. Hidell, H. Abrahamsson, B. Ahlgren, D. Li, P. Sjödin, V. Tanyingyong, and K. Xu, "Resource Management in Radio Access and IP-based Core Networks for IMT Advanced and Beyond", In *Science China Information Sciences*, vol. 56, no. 2, pp. 1-16, February 2013.
  DOI: 10.1007/s11432-012-4777-2
- **Paper B:** V. Tanyingyong, M. Hidell, and P. Sjödin, "Improving PC-based OpenFlow Switching Performance", In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'10)*, ACM, NY, USA, Article 13, 2 pages.
  DOI: 10.1145/1872007.1872023
- **Paper C:** V. Tanyingyong, M. Hidell, and P. Sjödin, "Using Hardware Classification to Improve PC-Based OpenFlow Switching", In *Proceedings of the 12th IEEE International Conference on High Performance Switching and Routing (HPSR), 2011*, pp. 215-221, 4-6 July 2011.
  DOI: 10.1109/HPSR.2011.5986029
- **Paper D:** V. Tanyingyong, M. Siraj Rathore, M. Hidell, and P. Sjödin, "Resilient Communication through Multihoming for Remote Healthcare Applications", In *Proceedings of IEEE Global Communications Conference (GLOBECOM), 2013*, pp. 1357-1363, 9-13 December 2013.
- **Paper E:** V. Tanyingyong, M. Hidell, and P. Sjödin, "Improving Performance in a Combined Router/Server", In *Proceedings of the 13th IEEE International*

Conference on High Performance Switching and Routing (HPSR), 2012, pp. 52-58, 24-27 June 2012. DOI: 10.1109/HPSR.2012.6260827

List of publications of the same author that are not included in this thesis:

- V. Tanyingyong, M. Hidell, and P. Sjödin, "Energy-Efficient Fixed Network Infrastructures: A Survey", In *Proceedings of 8th Swedish National Computer Networking Workshop (SNCNW 2012)*, pp. 21-25, 7-8 June 2012.
- V. Tanyingyong, R. Olsson, M. Hidell, P. Sjödin, and B. Pehrson, "Design and Implementation of an IoT-controlled DC-DC Converter", In *Proceedings of the third IFIP conference on Sustainable Internet and ICT for Sustainability (sustainIT), 2013*, pp. 1-4, 30-31 October 2013.

4.2 Contributions

Main Contributions

Our first major contribution [58] is to demonstrate, for the first time in the literature, that OpenFlow lookup can be offloaded using a COTS NIC. This leads to the next contribution [59], in which we design and implement an architecture for next-generation PC-based routers using OpenFlow switching as a forwarding engine. To improve the lookup performance of a PC-based OpenFlow switch, we introduce a fast data path based on caching of flow table entries in the on-board classification hardware on the NIC. This design is simple to implement yet offering significant throughput improvements when compared to regular software-based OpenFlow switching (over 40% and 30% higher throughput compared to Stanford's reference implementation and Open vSwitch's implementation respectively).

For enhanced reliability, our contribution [61] is that we investigate BFD as a component for redundancy with fast failover. We study the trade-offs between short reaction times for rerouting and low probability of false alarms. Our empirical evaluation shows that this combination is challenging to achieve under high traffic loads. Nonetheless, we propose a simple mechanism that provides fast failover and meanwhile maintains a very low probability of generating false alarms and unwanted rerouting decisions. This is accomplished by allocating dedicated system resources (a receive queue and a CPU core) for BFD control messages. According to the best of our knowledge, we are the first to carry out empirical studies with the focus on the performance of BFD running on a software router especially when affected by other traffic. Another contribution [60], which is in line with this solution, is that we extend our architecture for next-generation PC-based routers with OpenFlow support beyond solely routing. More specifically, we propose a suitable architecture for a combined router/server that integrates two main functional tasks–packet forwarding and server processing–into a single system. We utilize parallelism of the multi-core architecture to boost the performance. In addition, we devise a strategy to efficiently map packet forwarding and application processing

tasks onto the multi-core architecture on the PC-based router. In general, a combined router/server is useful in a scenario that requires both packet forwarding and application processing such as in data center and residential gateway. Thus, it is also applicable for our scenario in which we integrate BFD application process into a next-generation PC-based router that usually performs packet forwarding task.

Specification of the Author's Individual Contributions

Paper A is a summary of the most important results of a large Sino-Swedish research project. I am one of the co-authors in this article. My contribution is primarily in Section 4 of [57], in which I describe the overview of next-generation network architecture with flow-based networking.

Paper B, C, and E are my own original works under the supervision of my supervisors. I am the first author of these papers while my supervisors are co-authors.

Paper D is a joint work with my fellow PhD student, M. Siraj Rathore. He and I shared ideas, designed and performed the experiments, and had regular discussions throughout the work.

Chapter 5

Conclusions and Future Work

This chapter concludes the thesis work and outlines the directions for future work.

5.1 Conclusions

We demonstrate that a next-generation PC-based router with support for flow-based networking in an SDN context can be realized using open source software on COTS hardware components. By offloading the lookup from a CPU to a NIC, the overall performance is improved significantly with a negligible overhead.

We also devise a strategy to utilize a multi-core architecture on the PC-based router that enables it to perform other tasks beyond solely routing. More specifically, we propose a suitable architecture for a combined router/server that integrates two main functional tasks–packet forwarding and server processing–into a single system. Such a device is useful for scenarios such as in data centers in which each server acts as an end host as well as a relay host for other servers. We also study BFD as a component for redundancy with fast failover. We investigate the tradeoff between short reaction times for rerouting and low probability of false alarms. We demonstrate that this combination is challenging to achieve under high traffic loads. Nevertheless, we propose a simple mechanism that provides fast failover and meanwhile maintains a very low probability of generating false alarms and unwanted rerouting decisions.

Our work demonstrates the potentials of open router platforms for SDN. Our prototypes offer not only high performance with good reliability but also flexibility to adopt new software extensions. Such platforms will play a vital role in advancing towards the future Internet.

5.2 Future Work

Our work has opened up more possibilities on open router platforms as well as for network research in general. There are many interesting research directions we can

pursue, and we are currently interested in the area of energy-efficient networking. Thus, we aim to investigate challenges related to energy-efficient networking in an SDN context as follows.

Performance Trade-offs

In energy-efficient networking, a set of network devices and links are expected to be put in low-power or sleep mode to save energy while the traffic would be redirected to other network devices and links that are still active. In this scenario, the active nodes and links would need to handle higher traffic load. We plan to investigate the trade-offs between performance and power consumption in an SDN context. In addition, it is crucial to investigate how best to distribute the load among the active SDN switches to ensure that the packet forwarding in SDN can maintain high performance under such situations.

Reliability Trade-offs

Although the network can be pruned to save energy, the main trade-off is the network redundancy, which becomes an important challenge as the network connectivity is reduced. Moreover, the communication delay in the interaction between the control plane and the data plane of SDN could be increased due to the congestion since there are fewer paths that all switches in SDN can use to communicate with the controller. Further investigation is needed to find the right balance between reliability, delay, and power consumption in SDN.

Provisioning of Energy-Related Information

SDN, by design, has excellent support for introducing new control policies and services. However, it is still a challenge for SDN to realize energy-efficient networking because the underlying network devices normally do not provide any energy-related parameters that would be required by the centralized controller for making energy-efficient networking decisions. Although energy-related parameters might not be mandatory for some of the energy saving techniques, these parameters could still be added to further increase the total energy saving. In this aspect, the challenge is to find a novel and efficient way to provide the missing information regarding the power consumption of the network devices to the centralized controller.

Our Vision: Energy Control System

Our ultimate goal in this research area is to create an energy control system that monitors the traffic situation in the network and makes decisions to redirect communications and reconfigures the network to minimize the overall power consumption without sacrificing performance and reliability. This involves many components that also require further investigation. For example, research on suitable models

and policies for the energy control system, exploration of various aspects such as performance, reliability, security, scalability, optimization, and their trade-offs.

In any respect, we envision that the energy control system will be built around the following four components:

- *Flexible DC-DC converters* that are interconnected to form a small scale power grid.
- *A grid server* that coordinates the operations of the DC-DC converters and acts as a proxy to provide a communication interface for the centralized controller.
- *A centralized controller* that is a central control unit of the power control system. It has a global view of all grid servers and DC-DC converters in the network and makes decisions to redirect communications and reconfigures the network to minimize the overall power consumption based on real-time situations of the network.
- *An Energy Control Manager* that is a policy management unit. It is a policer that defines and governs the power distribution and usage in the network-wide SDN architecture.

Bibliography

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse through Virtualization. *Computer*, 38(4):34–41, 2005. ISSN 0018-9162.

[2] N. Beheshti and Ying Zhang. Fast failover for control traffic in Software-defined Networks. In *IEEE Global Communications Conference (GLOBECOM), 2012*, pages 2665–2670, 2012.

[3] A. Bianco, R. Birke, L. Giraudo, and M. Palacin. OpenFlow Switching: Data Plane Performance. In *IEEE International Conference on Communications (ICC), 2010*, pages 1 –5, May 2010.

[4] Big Switch Networks, Inc. Big Network Controller. `http://www.bigswitch.com/products/SDN-Controller`.

[5] Big Switch Networks, Inc. Big Switch Networks. `http://www.bigswitch.com/`.

[6] Z. Cai, A. L. Cox, and T. S. E. Ng. Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane. Technical Report TR11-07, Rice University, December 2011.

[7] Z. Cai and T. S. E. Ng. maestro-platform. `http://code.google.com/p/maestro-platform/`.

[8] Á. L. V. Caraguay, A. B. Peral, L. I. B. López, and L. J. G. Villalba. Software-Defined Networking: Evolution and Opportunities in the Development IoT Applications. *International Journal of Distributed Sensor Networks*, 2013.

[9] C. Chaudet and Y. Haddad. Wireless Software Defined Networks: Challenges and opportunities. In *IEEE International Conference on Microwaves, Communications, Antennas and Electronics Systems (COMCAS), 2013*, pages 1–5, 2013.

[10] Cisco Systems, Inc. DiffServ – The Scalable End-to-End QoS Model. White Paper, August 2005. URL `http://www.cisco.com/en/US/technologies/tk543/tk766/technologies_white_paper09186a00800a3e2f.pdf`.

[11] Cisco Systems, Inc. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018. White Paper, February 2014. URL http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf.

[12] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-Performance Networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):254–265, August 2011. ISSN 0146-4833. URL http://doi.acm.org/10.1145/2043164.2018466.

[13] A. Dixit, F. Hao, S. Mukherjee, T.V. Lakshman, and R. Kompella. Towards an Elastic Distributed SDN Controller. *SIGCOMM Comput. Commun. Rev.*, 43 (4):7–12, August 2013. ISSN 0146-4833. URL http://doi.acm.org/10.1145/2534169.2491193.

[14] D Erickson. What is Beacon? https://openflow.stanford.edu/display/Beacon/Home.

[15] D. Erickson. The Beacon Openflow Controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2178-5. URL http://doi.acm.org/10.1145/2491185.2491189.

[16] Ericsson. More than 50 billion connected devices. White Paper, February 2011. URL http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf.

[17] EZchip Technologies. NP-4 100-Gigabit Network Processor for Carrier Ethernet Applications Product Brief, 2011. URL http://www.ezchip.com/Images/pdf/NP-4_Short_Brief_online.pdf.

[18] K. Fall, G. Iannaccone, M. Manesh, S. Ratnasamy, K. Argyraki, M. Dobrescu, and N. Egi. RouteBricks: enabling general purpose network infrastructure. *SIGOPS Oper. Syst. Rev.*, 45:112–125, February 2011. ISSN 0163-5980. URL http://doi.acm.org/10.1145/1945023.1945037.

[19] A. Feldmann. Internet Clean-slate Design: What and Why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, July 2007. ISSN 0146-4833. URL http://doi.acm.org/10.1145/1273445.1273453.

[20] O. E. Ferkouss, I. Snaiki, O. Mounaouar, H. Dahmouni, R. Ben Ali, Y. Lemieux, and C. Omar. A 100gig network processor platform for openflow. In *Proceedings of the 7th International Conference on Network and Services Management*, CNSM '11, pages 286–289, Laxenburg, Austria, Austria, 2011. International Federation for Information Processing. ISBN 978-3-901882-44-9. URL http://dl.acm.org/citation.cfm?id=2147671.2147718.

[21] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, IETF, February 2012.

[22] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy. Flow processing and the rise of commodity network hardware. *SIGCOMM Comput. Commun. Rev.*, 39:20–26, March 2009. ISSN 0146-4833. URL `http://doi.acm.org/10.1145/1517480.1517484`.

[23] P. Gupta and N. McKeown. Algorithms for Packet Classification. *IEEE Network*, 15(2):24–32, 2001. ISSN 0890-8044.

[24] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated software router. *SIGCOMM Comput. Commun. Rev.*, 41:195–206, August 2010. ISSN 0146-4833. URL `http://doi.acm.org/10.1145/2043164.1851207`.

[25] IEEE. IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks. IEEE Standards 802.3az-2010, Institute of Electrical and Electronics Engineers (IEEE), October 2010. URL `http://www.ieee802.org/3/az/index.html`.

[26] D. Katz and D. Ward. Bidirectional Forwarding Detection (BFD). RFC 5880, IETF, June 2010.

[27] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Sköldström. Scalable fault management for OpenFlow. In *IEEE International Conference on Communications (ICC), 2012*, pages 6606–6610, 2012.

[28] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a Distributed Control Platform for Large-Scale Production Networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1924943.1924968`.

[29] M. Kuzniar, P. Peresini, N. Vasic, M. Canini, and D. Kostic. Automatic Failure Recovery for Software-Defined Networks. In *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), 2013*, 2013.

[30] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang. ServerSwitch: a programmable and high performance platform for data center networks. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 2–2, Berkeley, CA, USA, 2011. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1972457.1972460`.

[31] Y. Luo, P. Cascon, E. Murray, and J. Ortega. Accelerating OpenFlow Switching with Network Processors. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS

'09, pages 70–71, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-630-4. URL http://doi.acm.org/10.1145/1882486.1882504.

[32] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008. ISSN 0146-4833. URL http://doi.acm.org/10.1145/1355734.1355746.

[33] Microsoft. Virtual Private Networking: An Overview, September 2001. URL http://technet.microsoft.com/en-us/library/bb742566.aspx.

[34] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Implementing an OpenFlow switch on the NetFPGA platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 1–9, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-346-4. URL http://doi.acm.org/10.1145/1477942.1477944.

[35] NEC Corporation. NEC Global. http://www.nec.com.

[36] NEC Corporation. ProgrammableFlow Networking. http://www.nec.com/en/global/prod/pflow/images_documents/ProgrammableFlow_Brochure.pdf, 2011.

[37] NetFPGA. NetFPGA. http://netfpga.org/learn_more.html.

[38] Netronome. NFP-3200 Network Flow Processor, 2010. URL http://www.netronome.com/files/file/Product%20Briefs/NFP%20Product%20Brief%20(12-10).pdf.

[39] R. Neugebauer. Selective and Transparent Netronome Acceleration of OpenFlow Switches. White Paper, April 2013. URL http://netronome.com/files/file/whitepapers/Selective%20and%20Transparent%20Netronome-Acceleration%20of%20OpenFlow%20Switches%20Whitepaper_4-13.pdf.

[40] NOXRepo.org. About NOX. http://www.noxrepo.org/nox/about-nox/.

[41] NOXRepo.org. About POX. http://www.noxrepo.org/pox/about-pox/.

[42] Open Networking Foundation. OpenFlow. https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[43] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White Paper, April 2012. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[44] Open Networking Foundation. OpenFlow Switch Specification, Version 1.4.0 (Wire Protocol 0x05). Technical report, Open Networking Foundation, October 2013. URL `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf`.

[45] Open vSwitch. Open vSwitch – An Open Virtual Switch. `http://openvswitch.org/download/`.

[46] C. Perkins. IP Mobility Support for IPv4, Revised. RFC 5944, IETF, November 2010.

[47] Project Floodlight. Floodlight. `http://www.projectfloodlight.org/floodlight/`.

[48] L. Rizzo. netmap: a novel framework for fast packet I/O. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, USENIX ATC'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=2342821.2342830`.

[49] N. Sarrar, A. Feldmann, S. Uhlig, R. Sherwood, and X. Huan. FIBIUM: Towards Hardware Accelerated Software Routers. Technical report, Deutsche Telekom Laboratories, November 2010. Techical Report No 9.

[50] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. SDN Security: A Survey. In *IEEE SDN for Future Networks and Services (SDN4FNS), 2013*, pages 1–7, 2013.

[51] S. Sezer, S. Scott-Hayward, P.K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51 (7):36–43, 2013. ISSN 0163-6804.

[52] S.A. Shah, J. Faiz, M. Farooq, A. Shafi, and S.A. Mehdi. An architectural evaluation of SDN controllers. In *IEEE International Conference on Communications (ICC), 2013*, pages 3504–3508, 2013.

[53] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky. Advanced Study of SDN/OpenFlow controllers. In *the CEE-SECR '13: Central & Eastern European Software Engineering Conference in Russia*, October 2013.

[54] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. A Demonstration of Fast Failure Recovery in Software Defined Networking. In Thanasis Korakis, Michael Zink, and Maximilian Ott, editors, *Testbeds and Research Infrastructure. Development of Networks and Communities*, volume 44 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics*

and Telecommunications Engineering, pages 411–414. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-35575-2. URL http://dx.doi.org/10.1007/978-3-642-35576-9_46.

[55] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar. Carving Research Slices Out of Your Production Networks with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 40(1): 129–130, January 2010. ISSN 0146-4833. URL http://doi.acm.org/10.1145/1672308.1672333.

[56] R. Sherwood, G. Gibb, K.K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the Production Network Be the Testbed? In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1924943.1924969.

[57] G. Su, M. Hidell, H. Abrahamsson, B. Ahlgren, D. Li, P. Sjödin, V. Tanyingyong, and K. Xu. Resource management in radio access and IP-based core networks for IMT Advanced and Beyond. *Science China Information Sciences*, 56:169–184, 2013. ISSN 1674-733X. URL http://dx.doi.org/10.1007/s11432-012-4777-2.

[58] V. Tanyingyong, M. Hidell, and P. Sjödin. Improving PC-Based OpenFlow Switching Performance. In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '10, pages 13:1–13:2, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0379-8. URL http://doi.acm.org/10.1145/1872007.1872023.

[59] V. Tanyingyong, M. Hidell, and P. Sjödin. Using Hardware Classification to Improve PC-Based OpenFlow Switching. In *IEEE 12th International Conference on High Performance Switching and Routing (HPSR), 2011*, pages 215 –221, July 2011.

[60] V. Tanyingyong, M. Hidell, and P. Sjödin. Improving Performance in a Combined Router/Server. In *IEEE 13th International Conference on High Performance Switching and Routing (HPSR), 2012*, June 2012.

[61] V. Tanyingyong, M. S. Rathore, M. Hidell, and P. Sjödin. Resilient communication through multihoming for remote healthcare applications. In *IEEE Global Communications Conference (GLOBECOM), 2013*, pages 1357–1363, 2013.

[62] D. E. Taylor. Survey and Taxonomy of Packet Classification Techniques. *ACM Comput. Surv.*, 37(3):238–275, September 2005. ISSN 0360-0300. URL http://doi.acm.org/10.1145/1108956.1108958.

[63] The OpenFlow Switch Consortium. Openflow switching reference system. `http://archive.openflow.org/wp/downloads/`.

[64] A. Tootoonchian and Y. Ganjali. HyperFlow: A Distributed Control Plane for OpenFlow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1863133.1863136`.

[65] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On Controller Performance in Software-defined Networks. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=2228283.2228297`.

[66] Trema. Trema. `http://trema.github.io/trema/`.

[67] J.S. Turner and D.E. Taylor. Diversifying the Internet. In *IEEE Global Telecommunications Conference, 2005. GLOBECOM '05*, volume 2, pages 6 pp.–760, 2005.

[68] S.H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2):136–141, 2013. ISSN 0163-6804.

[69] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable Flow-Based Networking with DIFANE. *SIGCOMM Comput. Commun. Rev.*, 41(4):–, August 2010. ISSN 0146-4833. URL `http://dl.acm.org/citation.cfm?id=2043164.1851224`.

Part II

Research Papers