Doctoral Dissertation Academic Year 2014

Packet Scripting Architecture with Precise Packet Timing Control

Yohei Kuga

Graduate School of Media and Governance
Keio University
5322 Endo Fujisawa, Kanagawa, JAPAN 2520882

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

This dissertation proposes a scriptable packet processing architecture, "EtherPIPE", which is capable of precise timing controlled packet injection and precise timestamping.

With the diversification and performance of Internet services, the backbone of the Internet is growing steadily. Intelligent Network Interface Cards (NICs) of 10 Gbps bandwidth became a commodity and are widely deployed as server-to-server interconnections. Thus, packet processing controlled by software provides programming flexibility with high throughput, applicable for network troubleshooting, benchmarking, packet forwarding and Internet measurement.

Unfortunately, software packet processing cannot achieve precise timing control. Existing software processing is limited by the timer accuracy of the OS scheduler, and transmit and receive timing of packets are controlled at microsecond accuracy. Hardware packet processing can manage the timing to nanosecond accuracy. But network hardware devices are not programmable and supporting complex packet processing using them is difficult. Also, in a high-throughput network with more than 10 Gbps bandwidth, time resolution greatly affects the performance. Thus, timing-sensitive packet processing requires specialized hardware such as FPGA or ASIC.

EtherPIPE is a new architecture for a software packet processing designed for timing-sensitive network applications, which make use of the NIC hardware functions, its device driver and a programmable scripting framework. The EtherPIPE NIC consists of two functions to control packet receiving and transmission timing. The NIC receives a packet with a timestamp of PHY-clock accuracy and schedules the time of packet transmission as designated by scripts. The EtherPIPE packet I/O is abstracted as a character device and each packet data can be translated from a binary stream into an ASCII character string. By using this convention, we can program in the UNIX shell programming manner via character devices that can be connected by Stdin and Stdout.

As a result, the EtherPIPE architecture provides a flexible packet processing framework for applications that need a precise packet timing with the same precision as network special hardwares. To validate the effectiveness of our approach, we have developed a packet generator and a middlebox for latency emulation, and demonstrate that their timing is 100 times more accurate than software packet processing. Both applications required only a small number of lines of code. This architecture contributes the rapid and low-cost development of high-precision network features for network device prototyping and troubleshooting.

Keywords: Software-Defined Network, NIC hardware, Network I/O, Shell scripting, Internet Measurement.

Thesis Committee:

Supervisor:

Prof. Jun Murai, Keio University

Co-Adviser:

Prof. Hideyuki Tokuda, Keio University
Prof. Osamu Nakamura, Keio University
Dr. Kenjiro Cho, Director of IIJ Research Laboratory

要旨

本論文では，NIC にパケット送受信の時刻管理を持たせることにより，高精度かつ操作が柔軟なパケット処理が可能になる EtherPIPE アーキテクチャを提案する．

現在，インターネットサービスの高品質化と多様化に従って，バックボーンネットワークの広帯域化が進んでいる．サーバ間接続には，10Gbps を超えるネットワークインタフェースカード (NIC) が広く利用されるようになった．それにより，ソフトウェアによるパケット処理を用いることで，柔軟性と高スループットを両立したネットワークのベンチマークやトラブルシュート，パケット転送，ネットワーク計測を可能になった．

しかし，ソフトウェアによるパケット処理では，専用ハードウェアのような精密なパケット処理が難しいために，未だ専用のネットワークハードウェアが多く利用されている．既存のソフトウェア処理は，OS スケジューラのタイマー精度の限界によって，パケットの送受信タイミングはマイクロ秒精度で制御されている．ハードウェアでは，ナノ秒精度でパケットの送受信タイミングが管理できるため，ソフトウェアに比べて精密なネットワーク機器検証が可能である．一方で，ネットワークハードウェアでは，ソフトウェアのような柔軟なパケット処理内容の変更が難しく，特に，現在ソフトウェアでの開発が主流であるインターネット計測のような複雑な計測シナリオの実施が難しい．現在のソフトウェアパケット処理によるタイミング精度では，今後の広帯域化進むアプリケーションの性能を制限してしまう可能性がある．

EtherPIPE は，NIC ハードウェアによるパケット送受信時刻管理機能，デバイスドライバ，そして，パケット処理プログラミングフレームワークで構成される．EtherPIPE NIC は，ソフトウェアから制御可能な送信パケットの時刻管理機能と受信パケットのタイムスタンプ機能を，Ethernet PHY チップと同等の時間分解能で提供する．また，EtherPIPE デバイスドライバは，シリアルデバイスのようにシンプルにパケットデータを読み書きするために，汎用的なデバイスであるキャラクタデバイスとしてネットワーク I/O を抽象化する．それにより，Shell Script 環境上

で，既存のテキスト処理関数やOSコマンドを組み合わせることでパケット処理が可能になる．

本アーキテクチャを用いることで，専用ハードウェアと同等のナノ秒精度でのパケット処理アプリケーションを，ソフトウェアで開発可能になる．評価では，既存のソフトウェアに比べて，ネットワークテスタやミドルボックス機能を100倍以上の精度かつ，数行のネットワークスクリプティングで構築可能なことを示した．本アーキテクチャは，ソフトウェアのみの変更で高精度なネットワーク機能開発を可能にし，迅速かつ低コストでネットワーク機器評価やトラブルシューティングを行える新しいネットワーク計測基盤を提供することに貢献した．

キーワード: Software-Defined Network, NIC hardware, Network I/O, Shell scripting, インターネット計測.

Acknowledgments

I would like to thank to Prof. Jun Murai, my dissertation advisor, for giving me the direction of my research and letting me try to keep up my research life. His visions are always full of new challenges and encouraging me.

I also like to thank to Dr. Kenjiro Cho, my co-adviser, taught me how to research, step by step, from very beginning. I am extremely grateful to you for your support and the fun time spent with you. I would also like to thank my dissertation committee members: Prof. Hideyuki Tokuda and Prof. Osamu Nakamura for their helps for my research.

I also thank to Assoc. Prof. Rodney D. Van Meter III, Dr. Shigeya Suzuki, Prof. Akira Kato, Assoc. Prof. Hiroyuki Kusumoto, Prof. Keiko Okawa, Assoc. Prof. Keisuke Uehara, Assoc. Prof. Jin Mitsugi, Prof. Keiji Takeda, Dr. Jun Takei, Dr. Kenji Saito, Dr. Hideaki Yoshifuji, Dr. Noriyuki Shigechika, Dr. Hiroaki Hazeyama, Dr. Yasuhiro Ohara, Dr. Yojiro Uo, Dr. Keiichi Shima Dr. Hajime Tazaki, and Dr. Hirochika Asai for their precise advisory words to improve my research.

Thanks to all the member of Jun Murai Laboratory, especially Takeshi Matsuya, Masaki Minami, Katsuhiro Horiba, Kouji Okada, Masayoshi Mizutani, Michio Honda, Ryo Nakamura, Yukito Ueno and Mizuki Kimoto.

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Depending on diversification and quality of the Internet services, Internet backbone bandwidth is growing steadily. Network Interface Controller (NIC) throughput is growing over 10 Gbps and it widely used on the server-to-server interconnections. Current Operating System (OS) support multiple core packet processing, modern PC bus and memories and many offloading functions on intelligence NIC. Then, current software can run with the wide bandwidth packets with commodity PC. Software packet processing is utilized a lot of packet processing application with high throughput such as network troubleshooting, benchmarks, packet forwarding and Internet measurement.

Software packet processing is difficult to manage packet timing with high precision such as network hardwares. The precision of timing-sensitive applications is limited on software such as delay emulation, traffic shaping and latency measurement. The software is possible to control a complex packet process but the timing of packet processing is dependent on the OS timer interrupt. Usually, a OS timer resolution is set to 1 to 10 milliseconds, but, a hardware packet processing is controlled every few nanoseconds. The gap of software and hardware timer resolution significantly affects the performance of packet processing. It is important for the precise timing control as well as high throughput on the future software packet processing.

The packet processing required more precise timing developed with FPGA or ASIC based hardware. These purposefully designed hardware has a transmission buffer in order to control the accurate packet transmission time. So the network tester sends

the packets to a different API from the generic NIC, and it isn't available for general packet processing. Moreover, special network hardware is difficult to update the test scenarios and components. Current OS network stack has a lot of network functions and support protocols, but only hardware is difficult support various protocols.

EtherPIPE resolves the timing gap of the current software processing and special network hardware. All of packets generated by the software can be managed to send and receive time in EtherPIPE NIC hardware. Therefore, EtherPIPE is available on commodity server environment, and all traffic processed in software with flexible programming. Existing OS architecture cannot manage the only packet timing control of the microsecond-level. EtherPIPE enables control packet timing with the nanosecond-level. EtherPIPE enables both hardware precise timing control with software programmability.

The dissertation details NIC hardware design, device driver abstraction, software API and programming model on EtherPIPE architecture. On the implementation, EtherPIPE is developed on commercial 1G Ethernet FPGA NIC and its Linux device driver. In the result, we developed the nanosecond ping tool, delay emulation middlebox and other benchmark tools on chapter 6, and we confirmed that many timing-sensitive applications can be developed by EthPIPE framework with a small number of lines. The performance of device driver architecture supports 12.6 Mpps packet processing.

By the proposed architecture, many packet processing applications which need a precise packet timing such as benchmark equipments and middlebox can become to develop by software. In the future, we'll extend the EthPIPE architecture not only the packet processing applications but Also general purpose network applications such as transport protocol.

1.1 Contributions

The contributions of this dissertation are as follows:

Hardware-assisted Packet-Timing Control

Hardware-assisted Timing-Control architecture is a function of NIC hardware and it controls transmission timing of every packet data and managed a trans-

mission and receive timestamp for software packet processing. From the software side, all packets have EthPIPE header, and userspace applications rewrite the timestamp field data on EthPIPE header. The transmission function of NIC hardware watches the timestamp field of every packet and sends a packet at schedule time of transmission. The schedule time can be managed in parts per 8 nanosecond. Due to software applications assign the transmission timing, it's possible to program any timing-sensitive applications.

Ethernet Character Device

Ethernet Character Device is a network device for packet processing that can handle the high throughput and flexibility. Many network devices have been implemented as a special device to be accessed using the BSD Socket API. The BSD socket provides an interface to control packets with flexibility and good performance, however, we have to obey the programming paradigm of the BSD socket that forces us to write a long passage of the code for accessing raw packets. Ethernet Character Device abstract a network I/O of common character device. So applications access packet data only using read(2) / write(2) system calls on Ethernet Character Device API. And each packet data convert ASCII format, thus, to handle packets in the UNIX shell programming manner, input and output for packets must be expressed in character devices that can be connected by stdin and stdout. This dissertation proposes a scripting framework by Ethernet Character Device and EthPIPE, was compatible with the development flexible and easy for the software, the control packet processing of high-precision timing hardware.

1.2 Dissertation Outline

Chapter 2 introduces the research backgrounds and motivations, and Chapter 3 presents an overview of the proposed EtherPIPE architecture. Next, Chapter 4 explains the Hardware-assisted Packet-Timing control architecture and its API for software development and Chapter 5 explains a new packet scripting architecture which use shell script and UNIX command-line. Chapter 7 The overall evaluation of this dissertation

is shown in Chapter 6. Chapter 8 discusses related works on packet scripting and NIC hardwares. Finally, Chapter 9 concludes the dissertation.

Chapter 2

Backgrounds and Motivation

Testing the complex network is a really hard work. Server and network became visualization and interconnects are increasingly faster. To develop the network applications needs a stress test using common traffic with high-throughputs and a inter-interoperability test between network protocols.

The cost-effective servers enabled the network testing with high throughput by software packet processing. The test with complex scenarios needs the packet scripting on the high-performance servers. These software packet scripting supports a device development, protocol implementation, troubleshooting and measurement of the complex networks. The packet scripting is developed as a library of programming language or special command. In addition, the current Intelligence NIC has some hardware-assisted functions for high-bandwidth packet processing such as the checksum offload and TCP segment offload.

On the other hand, special hardware equipments are still used for measuring the details of network hardware. In particular, the verification of network hardware needs to measure the precise throughput, PPS (packet per second) and internal latency[9, 10].

The requirements of software network tester capabilities and performance of the software packet processing are as follows:

Throughput

The Ethernet 1G, 10G and 100G are widely deployed on the data center. Current server environment supports the intelligent NIC with 1G Ethernet (1GE) or 10G Ethernet (10GE) ports. The servers have a capability of 1GE/10GE line-rate

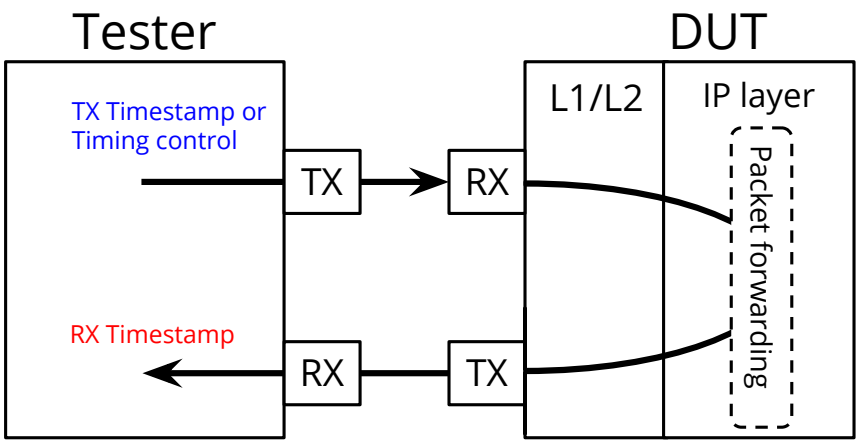

Figure 2.1: **RFC2544 Latency testing environment**

throughput on software processing.

Packet Per Second

Processing capacity of shortest packet (64 byte) with line-rate required 1.48 Mpps performance on 1GE interface, and the 10GE required 14.88 Mpps. For the software processing, this is the same meaning to process a packet in between 67.2 nanosecond and it means the 201 CPU cycle on 3 GHz of CPU. The handling the 10GE line-rate traffic on current OS network stack is difficult. But the software is capable of processing of the line-rate traffic by using libraries for packet manipulation[42, 15]. The frameworks bypass the OS network stacks and process the traffic by connecting directly with the network device.

Latency

The Fig 2.1 shows a simple network test environment. In this configuration, the Tester sends packets to the DUT (Device under test), and DUT send back the received packets to the Tester. The Tester needs two hardware features to measure the internal delay of the DUT; scheduling of transmit timing or transmit timestamp, and timestamping of the received packet.

This dissertation calls the timing of packet transmission and received timestamp control to packet timing. Also the required resolution of the measurement is in accordance with the internal clock source of the network device. Ethernet PHY is connected to the PHY and the internal system logic (MAC Layer) via the MII

(Media Independent Interface)[45]. The 1GE uses GMII bus interface which is operating as a clock source of 125 MHz (a clock means 8 nanosecond). And the 10GE uses XGMII interface and it operating at 156.25 with double words or 312.5 MHz (a clock means 6.4/3.2 nanosecond).

The detailed testing of 1GE / 10GE needs the both performance the potential of 1.48 Mpps / 14.88 Mpps packet processing and Ethernet PHY-level traffic control. Therefore, almost network tester is implemented in ASIC or FPGA hardware.

The hardware tester has different data manipulation API to the generic NIC. Traffic data and test scenario will be loaded with a specific buffer of the hardware tester by GUI frontend or special userspace API. The transmission logic sends packets from the buffer directly and tests scenarios build by setting the parameters. This architecture cannot be changed the flexible test scenarios like programming framework with software. The construction of the new test scenarios, there is a need to modify the firmware or hardware circuit logic.

2.1 Packet processing with Timing control

The performance of network measurement and analysis is dependent on the precisions of the transmission timing and reception timing of the packet. In order to implement the packet filtering and scheduling, OS manages the transmission and reception timing of packets in the kernel. The precision of packet transmission timing on software is greatly affected by the OS scheduler and the OS tick value for software interrupts. The default value of tick value on the Linux is set at the 1 4 millisecond (ms).Therefore, OS can only control packet timing in the order of milliseconds. But a hardware packet processing is controlled every few nanoseconds depend on the Ethernet MII clock source. In addition, timing of packet processing these might be further delayed depending on CPU utilization. In particular, measurement error in the delay measurement and end-node bandwidth estimation might increase at the burst packet processing.

Meanwhile, network measurement and performance analysis measure the elapsed time in the own application. The time precision may be improved by the use of hardware support functions such as CPU utilization counter. However, it packet timing control is function of OS kernel, so it's difficult to control from the processing applications. The OS need to offload the packet timing controls to support of timing in same

as hardware device.

2.2 Timing-sensitive packet processing

This section describes the timing-sensitive applications and its performance impact of timing precisions.

2.2.1 Network hardware benchmarking and prototyping

In many cases, network tester is designed by hardware for the performance measure of the network device. Because of various device testing is a necessary performance analysis of nanosecond accuracy and high-throughput. The hardware tester is provided in box type or NIC type of capture card. The box type tester is possible to test using a plurality of physical ports simultaneously. On the other hand, those of the PCIe card type are used for traffic analysis. The analysis device has a hardware assist function of packet sending and received timestamping. Also, the recent NIC has the function of IEEE1588v2 that is used for improving the accuracy of network time synchronization and packet time stamping. It's possible to correct the receive timestamp using Precision Time Protocol (PTP)[2] NIC hardware function. However, the PTP implementation on commodity NIC generates the timestamp on software, and the hardware timestamp is used for correction of the software value. To achieve the same accuracy of hardware value is necessary to generate timestamp on hardware.

2.2.2 Internet measurement

The timing of packet sending and receiving has a significant effect on a result of Internet measurement and analysis. The Internet structure is always evolving and growing, and the capturing the structural big-picture is difficult. Typically, we can measure and analyze the Internet properties by sending from the outside to the Internet hosts. Ping and traceroute[4] which are a measurement tools can estimate the distance between the target network structure from the outside. The possible time resolution of ping measurement is millisecond order. The RTT measurement with high accuracy can be expected to be the location and distance of Internet nodes.

For instance, some BitTorrent client guesses the distance and location of connected client by using ping-based RTT measurement for improving data-transfer throughput[43]. The precision of the RTT measurement is possible to improve of the estimated range of positions and distances of the BitTorrent node.

In other cases, Chapter 7 describes the measurement technique of Regional AS structure. The technique is combining the RTT and the path information by traceroute and ping which measured from the multiple locations and the result showed that the association between the logical network path information and the physical location on the Internet is possible. However, the existing ping and traceroute can only measure the RTT value with millisecond accuracy. In general, packet data goes about 197 kilometers in optical fiber at each millisecond. Therefore, ping-based measurement is possible to infer the continent-level and country-level Internet structure, but inferring the more detail such as city-level and PoP-level structure is difficult. The hardware-assisted ping is possible to measure the detail of geographic location of Internet hosts.

2.3 Summary

This chapter presented the needs of new packet scripting architecture which combines software and hardware approach. Currently, applications that require packet timing control with high accuracy are implemented as a special network hardware by ASIC/FPGA. The software packet processing is possible to control a complex tasks, but its time resolution is dependent on the OS timer interrupts. Developing the software packet processing with high precision is necessary to use both methods with hardware timing control and software programmability. The next chapter gives an overview of EtherPIPE precise packet scripting architecture.

Chapter 3

EtherPIPE Overview

The network hardware is capable of high-precision packet timing management but is difficult to circuit update of network applications. EtherPIPE is a new architecture for a software packet processing designed for timing-sensitive network applications. This chapter explains the approach of EtherPIPE how to manage the packet timing on NIC hardware.

3.1 Basics

The packet timing on software processing is managed on the scheduler of operating system. Thus, the timing accuracy is difficult to achieve the same precision of network hardware.

The packet timing can be expected high accuracy by controlling on hardware. This section discusses the abstraction and API of the packet timing that can support packet processing applications.

The EtherPIPE constructed from the following two functions:

1. Hardware-assisted Timing-Control: functions of the scheduled packet transmission and received timestamp on NIC hardware
2. Ethernet Character device: a network device for shell-based packet scripting with high-throughput and flexible programming

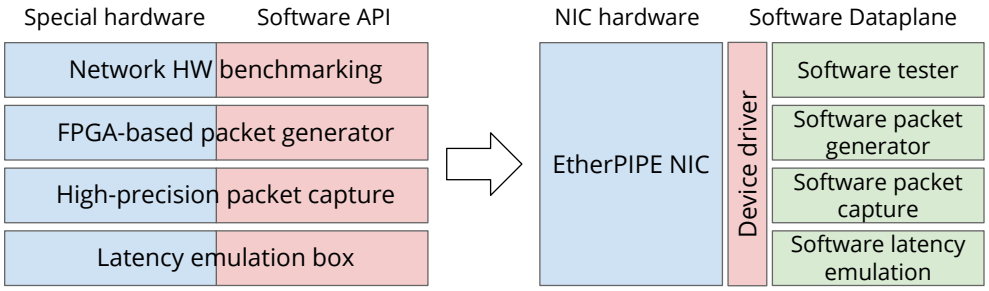

Figure 3.1: **EtherPIPE NIC manages the transmission timing on the hardware. And all of the network applications could be developed the same hardware structure.**

The EtherPIPE NIC gives the header of the transmission timing for all packets for software processing. In this design, software does not manage the packet timing with an OS scheduler, to control the transmission time by updating the value of the timestamp header. The EtherPIPE NIC observes the value of timestamp header and waits to send a packet until the scheduled timing.

Figure 3.1 shows the comparison of EtherPIPE design and the existing network hardware. The hardware is consisted of special hardware logic and its software API. This approach is difficult to change the applications of hardware logics.

Meanwhile, EtherPIPE architecture is possible to change the packet timing by using software. All of the network applications could be developed the same hardware structure.

3.2 Target applications

Figure 3.2 shows target of EtherPIPE architecture; Middlebox and End host. This is different method to manage the packet timing. The Middlebox such as delay emulator and traffic shaper uses the received timing and transmission timing. And the End host such as traffic generator and high accuracy ping uses process own timing and packet received timing.

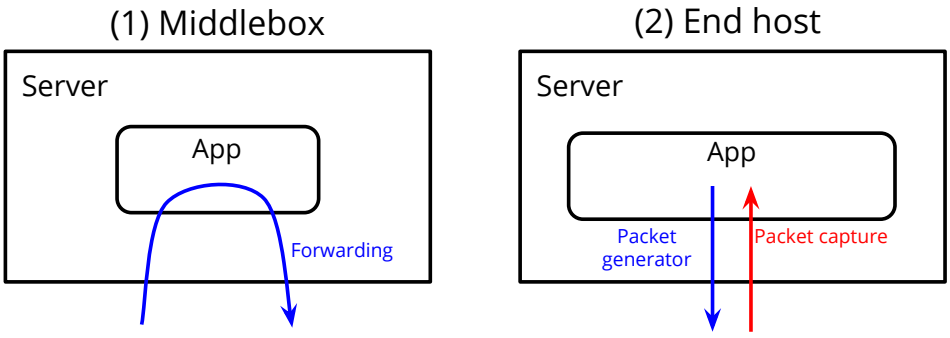

Figure 3.2: **Target applications of EtherPIPE**

3.3 System design

Fig 3.3 show EtherPIPE architecture overview. EtherPIPE is constructed of two functions; hardware assisted Timing Control architecture and shell script-based packet processing architecture.

The hardware assisted Timing Control architecture is a function of NIC hardware and it control transmission timing of every packet data and managed a transmission and receive timestamp for software packet processing. This architecture is constructed of two hardware logics; Received timestamp and scheduled transmission. From software side, all packets has EthPIPE header, and userspace applications rewrite the timestamp field data on EthPIPE header. The transmission function on NIC hardware watch the timestamp filed of every packets and send a packet at schedule time of transmission. The received function on NIC hardware add the received timetstamp value from NIC conter logic to EtherPIPE timestamp header field.

Next, shell script-based packet processing architecture supports the high throughput processing and flexibility deployment. This architecture use specific device I/O, Pktdev, which is common character device for packet processing. Many network devices have been implemented as a special device to be accessed using the BSD Socket API. The BSD socket provides an interface to control packets with flexibility and good performance, however, we have to obey the programming paradigm of the BSD socket that forces us to write a long passage of code for accessing raw packets. Pktdev abstract a network I/O to common character device. So applications access packet data only using read(2) / write(2) system calls on pktdev API. And each packet data convert ASCII format, thus, to handle packets in the UNIX shell programming manner, input

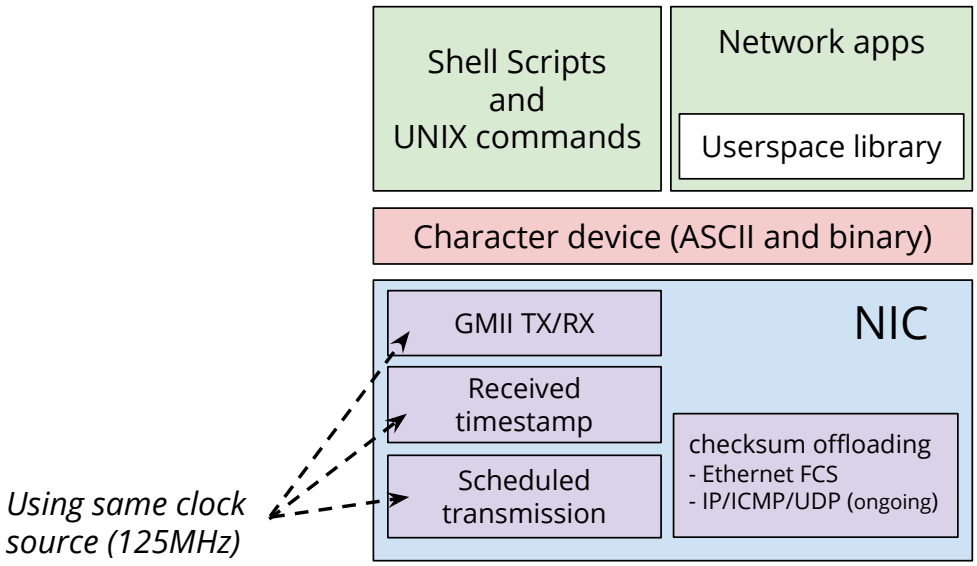

Figure 3.3: **EtherPIPE network scripting architecture**

and output for packets must be expressed in character devices that can be connected by stdin and stdout. In this paper, I propose a scripting framework by Pktdev and EthPIPE, was compatible with the development flexible and easy by the software, the control packet processing of high-precision timing hardware.

3.4 Comparison with the existing approach

Fig 3.4 show the comparison with EtherPIPE timing controlled architecture and existing methods as network hardware tester and sofware packet processing. EtherPIPE manage the packet timing by using EtherPIPE header. EtherPIPE NIC looks and rewrites the header value.

EtherPIPE NIC add timestamp value when received packets. Other method, software packet processing, NIC doesn't add timestamp value for each packets. Userspace application such as tcpdump[46] and Wireshark[49] add timestamp value on own processing application. In this software method, it's difficult to generate precise timestamp value because timestamp is depend on software time resolutions.

And EterPIPE control the packet sending time by using EtherPIPE header value. The based timing is generated from same counter logic on NIC. Meanwhile, software

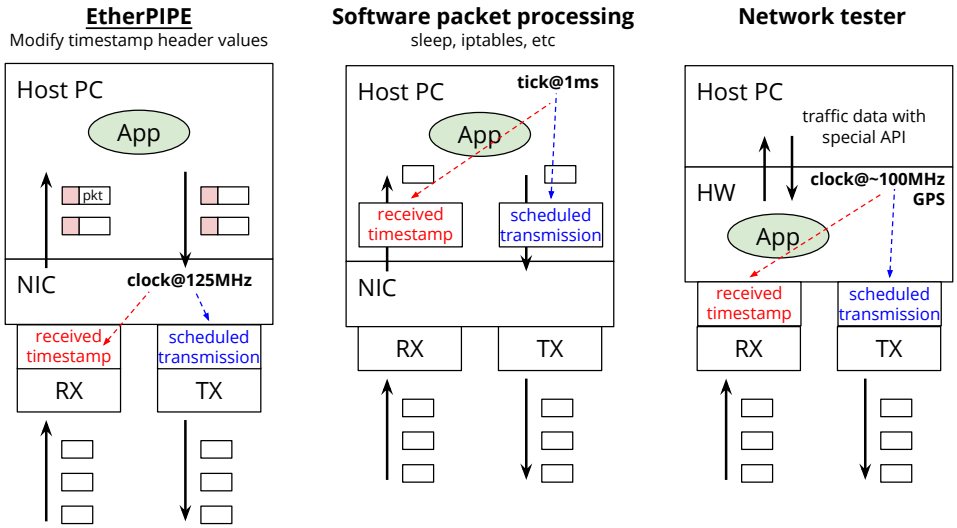

Figure 3.4: **Comparison of EtherPIPE with existing methods**

approach control sending packet timing on OS kernel. Using OS software packet processing, user can select any timing control algorithm which implemented on OS. But this timing resolution is depend on software timer resolution.

3.5 Summary

In this chapter, we proposed a new packet scripting architecture for a software packet processing designed for timing-sensitive network applications. EtherPIPE constructs the two functions of the scheduled transmission and the received timestamping. The next chapter describes the details of NIC hardware design and implementation. And Chapter 5 explains the design and implementation of the device driver and the programming framework which use shell script and UNIX command-line.

Chapter 4

Hardware assisted Packet Timing Control

This chapter introduces the EtherPIPE NIC architecture and implementation with FPGA-based NIC prototype.

4.1 Overview

The EtherPIPE NIC presents hardware assisted receive time stamping and send timing control with 8 ns precision. The system can be controlled by simple shell scripts using

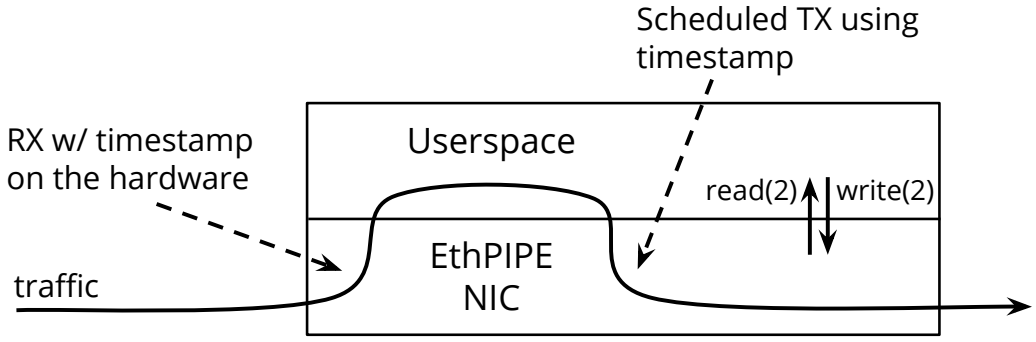

Figure 4.1: **EtherPIPE NIC**

the Ethernet character device.

The performance of a software dataplane using a commodity NIC is increasing and we can more easily build network applications having both the high performance and programmable flexibility. However, it's difficult to build applications which require packet sending with strict latency control, such as network device benchmarking and delay emulation with high accuracy. Therefore, these network applications require an ASIC or FPGA logic as single purpose hardware.

The EtherPIPE NIC is a new architecture for a software dataplane for timing-sensitive network applications and which consists of an FPGA-based NIC and device driver [Fig 4.1]. This NIC implementation has two offloading functions to control packet input and output timing. The NIC receive a packet with a timestamp of PHY-clock accuracy and schedules the time of packet transmission as designated by userspace scripts.

4.2 Timestamp accuracy

EtherPIPE NIC targeted value of controllable receive and send timing is the same as the PHY chip. The current NIC implementation is on Lattice Inc's FPGA development board which has two 1GE ports and a PCI express interface[30]. The NIC defined a 48 bit counter on the FPGA-based NIC, and send, receive, and timestamp logic use the same 125 MHz clock source. Each received packet has the value from the 48 bit counter and the scripts on userspace can schedule the timing of transmission to compare the designated value by scripts and the counter value.

4.3 Timestamp format

The timestamp format is different between packet forwarding and packet generation. When building packet forwarding applications such as delay emulation, the transmission timing should define the difference from the received hardware timestamp. Meanwhile, when programming a packet generator such as a network hardware benchmark tool should be represented by a difference from the first packet. The current NIC implementation supports both timestamp formats and applications can choose the format.

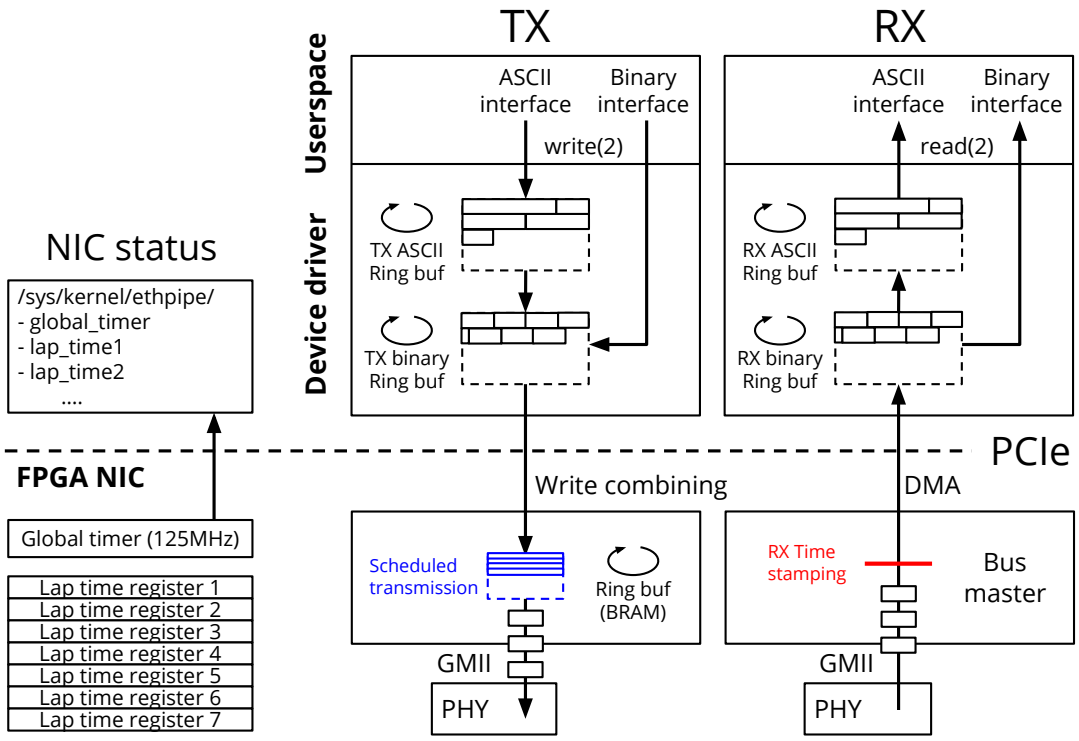

Figure 4.2: **Implementation of TX-Timing offload NIC**

4.4 Implementation

4.4.1 FPGA NIC hardware

Figure 4.2 shows an overview of EtherPIPE implementation. This is developed an implementation of Ethernet Character Device for a commodity FPGA network card on Linux, and implemented almost all of the primitive functions.

The FPGA implementation supports basic transmit and receive function for 1000BASE-T and hardware offloading functions such as hardware timestamp and five-tuple hash of the Ethernet Character Device Raw interface. The NIC design of the FPGA logic supports ring buffers for transmitting and receiving packets, and the bus master mode to transfer data between the NIC and the main memory.

The EtherPIPE NIC was developed the FPGA logic using the LatticeECP3 versa

development kit[30] by Lattice Semiconductor Inc. The Lattice ECP3 versa kit has two 1000BASE-T interfaces and one PCI Express interface, therefore, it can be used to test the forwarding case by Ethernet Character Device network scripting.

The receive logic is the same design as an ordinary NIC. The FPGA becomes a bus master and writes packet data directly to the host RX Raw ring buffer.

The transmission logic used PCI Express PIO (Programmed I/O) write. The device driver writes packet data to the FPGA using PCIe with write combining. The prior experiments showed that normal PCI Express PIO writing performs at 40 MB/s but PIO writing with PCIe write combining performs at about 175 MB/s, or over 1Gbps.

We developed the FPGA logic with only a 32KB ring buffer in order to simplify the circuit. The Data format of the Raw interface has the frame size of the packet in the head of the data, so the FPGA TX logic can find the data boundary. In the design of TX logic, the device driver doesn't need to update the write pointer of the FPGA ring buffer with respect to writing each packet. The device driver only updates the write pointer when it has finished all of the data in the TX Raw ring buffer in the device driver.

4.4.2 Device driver

The Ethernet Character Device device driver has two 1MB ring buffers for Raw and Shell interface used for the sending and receiving device of each port. The Ethernet Character Device is a general character device, so userspace applications can send and receive packet data with any buffer size of **write(2)/read(2)** system call. The buffer size of **write(2)/read(2)** depends on the implementation of user commands.

When sending a packet using the Shell interface, the device driver gets the packet data from userspace and writes it to the TX ASCII ring buffer in the device driver. Next, the device driver converts the ASCII packet data in the TX ASCII ring buffer to the binary format of the Raw interface, and writes to the TX Raw ring buffer. Finally, the device driver reads the binary packet data from the TX Raw ring buffer and writes the packet data to the FPGA sending buffer using **memcpy(3)**. When it finishes writing the packet data, the device driver updates the write pointer of the FPGA sending slot to the frame size. If the sending ring buffer still has data, the device driver repeats the sending process.

When sending a packet using the Raw interface, the device driver writes the binary

data to the TX Raw ring buffer directly.

Chapter 5

Ethernet Character Device

This chapter proposes the Ethernet Character Device which is new network scripting framework[28, 29].

In order to realize Software-Defined Networking (SDN), many APIs and libraries have been proposed, such as OpenFlow and its controller[36, 37, 50], and API for Commercial Network Device[6, 24]. If we could use the UNIX shell commands for network processing, the scripting environment isn't depend the software libraries, and we think it become new lightweight network processing tools.

However, current network devices on a UNIX-like OS are implemented as special device with a specific API, and can not be used in an shell environment. One of the reasons why network I/O is usually implemented as a network device rather than a character device is the need to access different layers in a packet such as the Ethernet frame, the IP packet and the UDP/TCP datagram.

The BSD socket is one abstraction allowing network devices to access raw packets, and behaves as glue or a buffer to read/write a payload as characters. The BSD socket provides an interface to control packets with flexibility and good performance, however, applications have to obey the programming paradigm of the BSD socket that forces us to write a long passage of code for accessing raw packets. Of course, applications can inject packet streams more easily by using a virtual network device[26] or one of several libraries to access raw sockets[13]. These APIs provide easy access to raw socket but still require programming specific to networking. Following the characteristics and friendliness of these APIs, we would like to provide a simple network

I/O for network processing on an OS in the same manner of device files for storage devices.

Ethernet Character Device provides a character device interface for a network device. A network device on Ethernet Character Device is abstracted as a device file on an OS, and packets on the network device are transformed to files or streams on the device file. Ethernet Character Device also serves as truly simple input/output functions for scripting packets like file processing on a UNIX command line. In Ethernet Character Device's network scripting, various packet processing can be achieved by I/O redirection through standard input (<), standard output (>) and pipe (|). We believe that the Ethernet Character Device network scripting framework brings a more flexible/lightweight programming paradigm that allows us to develop a packet processing application for a SDN.

Many features of the network have been fixed in hardware until the advent of SDN. With the advent of SDN, we can now develop new network functions quickly and flexibly using a commodity PC and its software.

To provide further flexibility to software-based development in networking, we developed Ethernet Character Device as an Ethernet device driver for a commodity FPGA network card on Linux. Our software and FPGA circuit are available as open source[2]. Combining Ethernet Character Device with hardware offloading functions of the FPGA or other network processors, more powerful network scripting or network processing can be achieved.

5.1 Programing model

5.1.1 Primitive Functions for Packet Processing

As a network scripting framework, primitive network processing functions should be provided by Ethernet Character Device as commands on an OS. Before designing the Ethernet Character Device network scripting framework, we explore the primitive functions needed to program network applications as shell scripts, and try to define a primitive function set.

We focus on network applications on the data-link layer as a first step. Note that we

[2]Ethernet Character Device, `https://github.com/sora/pktdev`

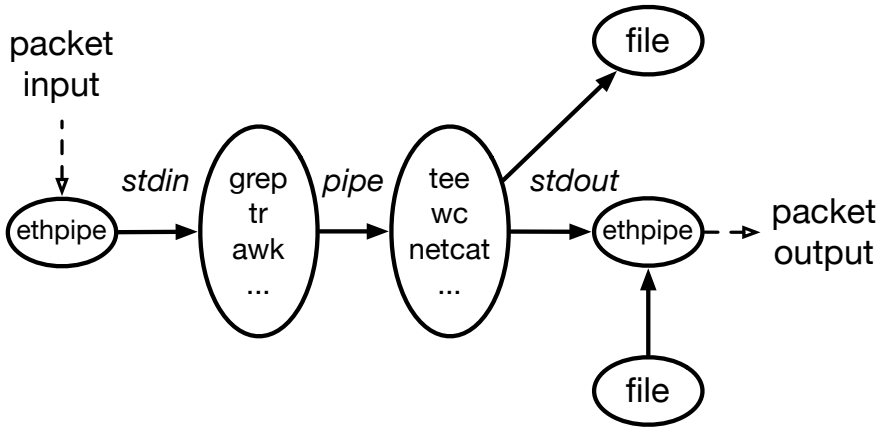

Figure 5.1: **Network scripting**

refer to Ethernet frames, IP packets and/or TCP/UDP datagrams, all as `packets'' in the following sentences. We use the terms`Ethernet frames'', `IP packets'' and`TCP/UDP datagrams" when we would like to distinguish the data format on each layer.

Typical network applications on the data-link layer are as follows, network test and diagnosis tools such as packet generators and packet capture, and network elements such as Ethernet switch and tunnel gateways. These applications can be composed of five primitive functions 1) packet generation (packet sending), 2) packet capturing (packet receiving), 3) forwarding, 4) packet filtering, and 5) header modification.

5.1.2 Packet Sending and Receiving

All network equipments minimally require the functions of packet sending and/or packet receiving. TCP connections normally employ both packet sending and packet receiving. An application uses **read(2)** and **write(2)** on a socket file descriptor for a TCP connection.

Packet capture tools such as tcpdump[46] or WireShark[49] require only the packet receiving function. On the other hand, packet generator applications, such as pktgen[40] or scapy[8], are mainly composed of the packet sending function.

OpenFlow defines *Packet-in* and *Packet-out* functions[48]. Various OpenFlow libraries provide *Packet-in* and *Packet-out* APIs. Using these APIs, an open flow controller can send and receive arbitrary packets from OpenFlow switches.

Table 5.1: **Translate functions from packet processing to Ethernet character device**

Packet processing	Ethernet character device
receive packets	read from input device
send packets	write to output device
forward	copy from inputs to outputs
filter	search data pattern
modify headers	translate characters

5.1.3 Filtering

Filtering function is required by a firewall, port mirroring or network injector. Berkeley Packet Filter (BPF)[35], OpenBSD Packet Filter (PF)[21] or IPFW[31] are supported in various BSD Operating Systems.

5.1.4 Forwarding

Repeaters, switches and routers require a forwarding function to interconnect an input port and an output port. Several OSes support the forwarding function in the kernel. Recently, Linux **brctl(8)** and Open vSwitch[7] provide more flexible forwarding control.

5.1.5 Header Modification

Header modification is a key function to achieve forwarding, routing or encapsulation. Filtering tools on various OSes or *Flow-Mod* action of Open vSwitch enable users to modify protocol headers. Usually, a header modification function is hidden in the kernel space. The RUMP (Runnable Userspace Meta Program) kernel of NetBSD[5] provides a header modification environment in the user space.

5.2 Network scripting

We define "network scripting" to be processing packets in the UNIX shell interface or shell programming. In this section, we explain the design of the Ethernet Character Device network scripting framework to achieve the basic functions for network processing mentioned in Section 5.1.1 on the UNIX shell interface.

Table 5.1 shows the correspondence between basic functions and commands on the UNIX shell interface. In the UNIX shell programming framework, an application can be written by a chain of device files, files, and commands, concatenated by redirection expressions, using stdin (<), stdout (>) and pipe (|).

To handle packets in the UNIX shell programming manner, input and output for packets must be expressed in character devices that can be connected by stdin and stdout. Character device type network I/O uses the **read(2)** system call to a device file for packet receiving, and the **write(2)** system call to send packets to the device file.

The forwarding function can be simply achieved by copying data from the output of a device file to the input of the another device file. Also, redirection and concatenating commands will provide forwarding packets to multiple ports.

Each packet is a simple string, therefore, a combination of **grep(1)** and **tr(1)** will give a filtering function and a header modification function. Using regular expressions in **grep(1)** or **sed(1)**, we will describe a complex search pattern in a line shell script.

Figure 5.1 shows an overview of network scripting. Concatenating character device type network I/O and shell commands, we can process packets as files in a UNIX shell interface. Of course, the network scripting inherits the UNIX shell programming paradigm, so a custom network scripting command can be reused in another network script. Through our network scripting, interactive processing against network streams can be realized.

5.3 Design of Device formats

The Ethernet Character Device device tree and its name space are shown in Figure 5.2. For simplicity, we consider the case where only one multi-port network card is inserted into a computer. We locate a network device in `/dev` as is the case with other devices. Therefore, Ethernet Character Device provides abstracted character device files under `/dev/ethpipe/`. Each physical port on a network card is labeled as an independent

```
/dev/
  |-- /ethpipe/
        |-- 0        # Shell IF port 0
        |-- 1        # Shell IF port 1
        |-- r0       # Raw IF port 0
        |-- r1       # Raw IF port 1
```

Figure 5.2: **Ethernet Character Device device names**

```
[TIMESTAMP] [DST_MAC] [SRC_MAC] [Eth_Type] XX XX XX XX ...
[TIMESTAMP] [DST_MAC] [SRC_MAC] [Eth_Type] XX XX XX XX ...
[TIMESTAMP] [DST_MAC] [SRC_MAC] [Eth_Type] XX XX XX XX ...
```

Figure 5.3: **Shell interface format**

device such as `/dev/ethpipe/0` or `/dev/ethpipe/1`.

Ethernet Character Device creates two device interfaces at each physical port; one is the **shell interface** and the other is the **raw interface**. The shell interface is designed to access packets by using ASCII for network processing on a shell. The device name is described by only port number in `/dev/ethpipe/`.

The raw interface enables to access packets in a binary format for high-bandwidth network processing. The device name on the raw interface is labeled with **'r'** + port number. For instance, physical port **'0'** and **'1'** can be described as in Figure 5.2.

5.3.1 Shell interface

The shell interface is used for network scripting in the shell. Figure 5.3 presents the format of the shell interface. Packets are presented in ASCII, one packet per line and Ethernet header fields and payload in hexadecimal notation are separated with space characters by the kernel driver. Using the shell interface, we can parse packets by traditional command-line tools.

This shell interface on Ethernet Character Device is simple, however, two alternatives for the ASCII expression are considered. One is expressing MAC addresses and IP Addresses in ASCII, the other is expressing all protocol headers in ASCII. Of course,

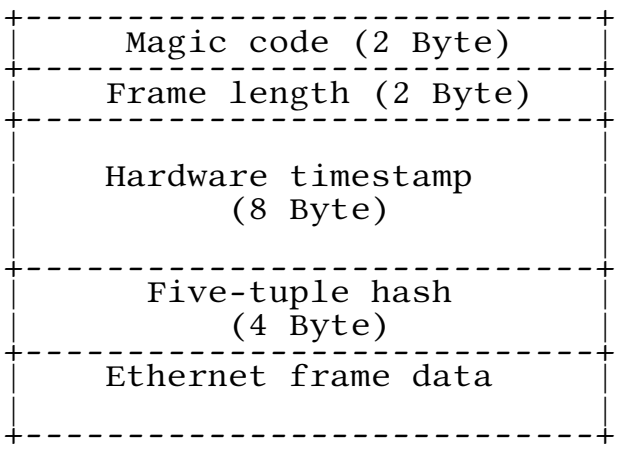

Figure 5.4: **Raw interface format**

adopting these expressions on Ethernet Character Device will give more control to network scripting, these expressions sacrifice processing time, data size and overhead on kernel drivers. Considering these trade-offs on ASCII expression, we take a simple ASCII expression described in Figure 5.3.

5.3.2 Raw Interface

The raw interface is used for network processing in high-bandwidth network connections. Figure 5.4 shows the format of the raw interface. This interface has metadata that indicates a hardware timestamp, a frame length, a five-tuple hash and Ethernet frame data. All Ethernet Character Device metadata are computed by hardware, and can be used by network processing software in the user space.

The hardware timestamp is described in a 64-bit counter value with 8 nanoseconds resolution taken when the head of a packet arrives at the network hardware. Wireshark[49] and its capture format PcapNG[17] support a nanosecond timestamp with NIC hardware counters. We use the same 64-bit timestamp by converting the Ethernet Character Device raw format to the PcapNG format.

The five-tuple hash is a hash value of *{source IP address, destination IP address, protocol number, source port number, and destination port number}* for the high throughput packet processing. Network processing often uses five-tuple hash values for identify-

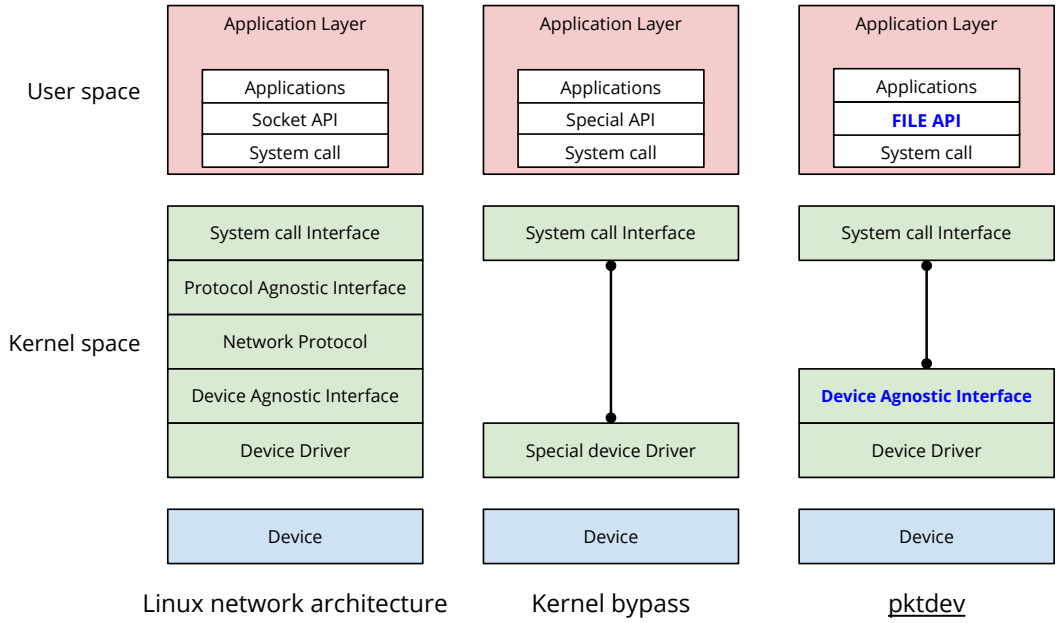

Figure 5.5: **Comparison of Ethernet Character Device and existing network scripting architecture**

ing unique IP flows on routers, firewalls and load-balancers.

5.4 Implementation

Figure 5.5 and 5.6 compare the Ethernet Character Device and existing network scripting architecture. Ethernet Character Device device driver only focuses the packet processing. So the device driver bypasses protocol layer functions. The device driver directly use Linux NIC transmission API as *ndo_start_xmit*.

Figure 5.8 and 5.7 show the high-level description of transmission implementation. Ethernet Character Device uses multiple CPU core for packet transmission and each kernel thread directly send packets to NIC driver.

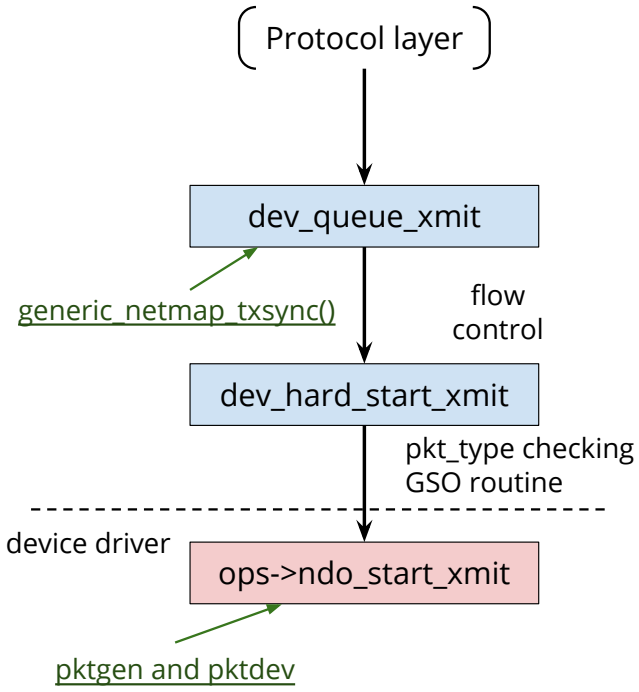

Figure 5.6: **Ethernet Character Device device directly use Linux NIC transmission API as ndo_start_xmit**

5.5 Applications

This section shows examples of network scripting by Ethernet Character Device. Mainly, we explain the examples of primitive functions mentioned in Section 5.1.1.

5.5.1 Packet capture and generation

Command 1: packet generation

```
$ cat packet.dump > /dev/ethpipe/r1
```

Command 2: packet capture

```
$ cat /dev/ethpipe/r0 > packet.dump
```

Command 3: decapsulating Ethernet header and store IP packets

```
$ cut -d' ' -f4- /dev/ethpipe/0 > ip-packets.dump
```

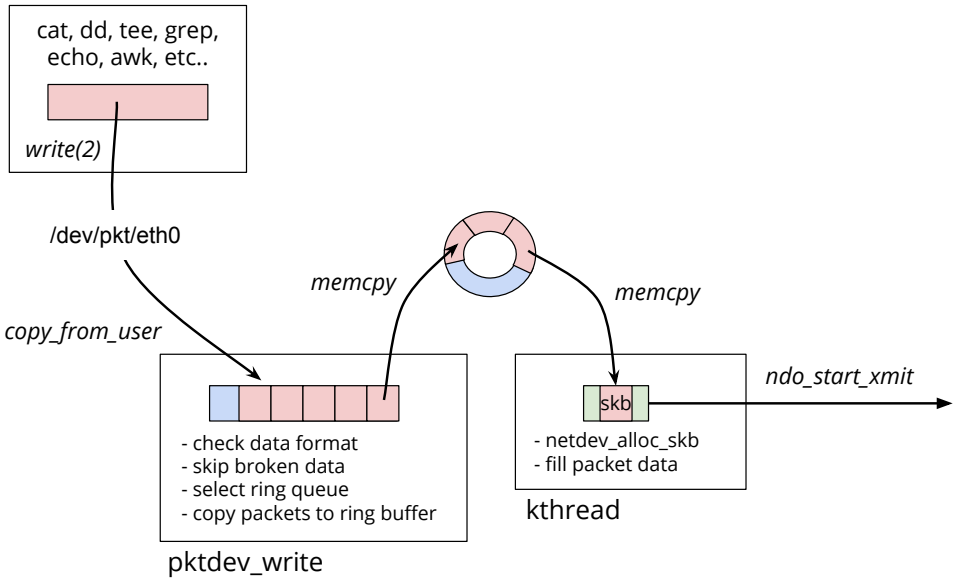

Figure 5.7: **Ethernet Character Device TX overview: the device driver use kthread for packet transmission**

Command 4: packet capture with PcapNG format

```
$ ethdump < /dev/ethpipe/r0 > dump.pcapng
```

Command 5: Port mirroring

```
$ cat /dev/ethpipe/0 \
  | tee /dev/ethpipe/0 > /dev/ethpipe/1
```

Packet monitoring or packet analysis often employs **tcpdump(1)** or WireShark to store packets in Pcap format or PcapNG format files. Ethernet Character Device is suited to capture packets or to generate packets from both the Raw and Shell interfaces.

Command 1 shows an example to generate (replay) packets from the Raw interface. Simply reading a Raw format file by **cat(1)** and redirecting stdout to the raw interface to Port~1, packets will be sent through Port~1.

Command 2 describes packet capturing via shell scripting. In contrast to Command 1, the packet capturing scripts redirects the raw interface to a file.

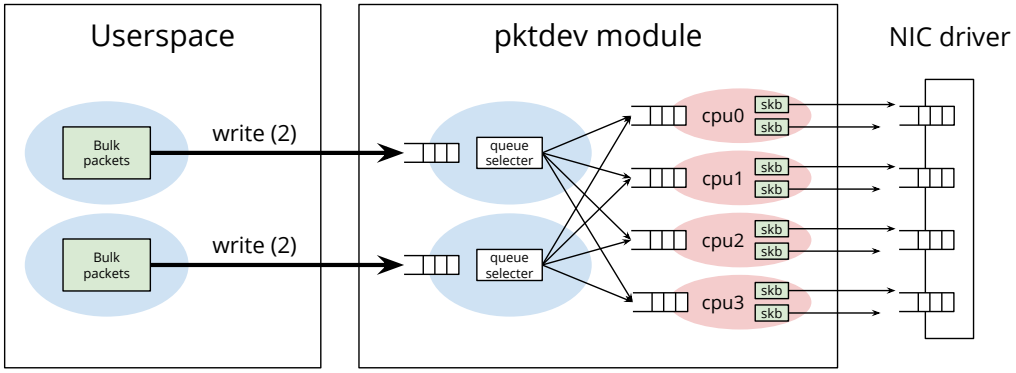

Figure 5.8: **Ethernet Character Device TX overview: sending packets by multiple CPU core**

Command 3 removes the Ethernet header of each packet from Port~0 and stores the IP packets into a file by redirection of stdout. Because of the shell interface of the character device (`/dev/ethpipe/0`), Ethernet Character Device enables **cut(1)** to separate the Ethernet header from a packet.

ethdump in Command 4 is the original shell command to store packets in the PcapNG format with hardware offloading of a FPGA network card. Through **ethdump**, a nanosecond-accuracy timestamp will be contained in each PcapNG format packet.

Port mirroring can be composed of chains of **tee(1)** and the shell interfaces of Ethernet Character Device like Command 5. The example of Command 5 mirrors received packets from Port~0 to Port~0 and Port 1. Adding a set of **tee(1)** and pipe (|), the number of the destination ports can be extended.

5.5.2 Mac address filtering

Command 6: filtering dstmac

```
$ gawk '$1=="001122334455"{print $0}' \
  < /dev/ethpipe/0 > /dev/ethpipe/1
```

Command 6 forwards packets from Port 0 to Port 1 on the network interface card only when the destination MAC address of a packet matches **00:11:22:33:44:55** by **gawk(1)**.

5.5.3 Decapsulation and Encapsulation

Command 7: VLAN tagging

```
$ sed -e 's/^\([^ ]* \)\{2\}/&8100 00 01 /' \
  < /dev/ethpipe/0 > /dev/ethpipe/1
```

Command 8: VLAN untagging

```
$ sed -e 's/8100 00 01 //' \
  < /dev/ethpipe/0 > /dev/ethpipe/1
```

Command 9: VLAN translation

```
$ sed -e 's/8100 00 02 /8100 00 01 /' \
  < /dev/ethpipe/0 > /dev/ethpipe/1
```

Command 3 decapsulates the Ethernet header from a packet. An 802.1Q VLAN operation can be described by **sed(1)**; VLAN tagging, VLAN untagging and VLAN translation are depicted in Command 7, 8 and 9 respectively.

5.5.4 Overlay tunneling

Command 10: L2 over TCP tunneling

```
[192.168.0.1] $ nc -l 9999 < /dev/ethpipe/0 \
                > /dev/ethpipe/0
[10.0.0.1] $ nc 192.168.0.1 9999 \
             < /dev/ethpipe/0 > /dev/ethpipe/0
```

Command 11: ssh tunneling

```
$ cat /dev/ethpipe/r0 \
  | ssh sample.com "cat >/dev/ethpipe/r0"
```

Several types of overlay tunneling can be described in Ethernet Character Device. In Command 10, L2 over TCP tunnel is achieved by **nc(1)**. **nc(1)** of the node `192.168.0.1` (the first line) reads packets from Port~0 and encapsulates read packets

into a TCP (port 9999). In the second line, node `10.0.0.1` connects stream to `192.168.0.1` TCP port 9999, decapsulates packets, and throws decapsulated packets into port 0 of node `10.0.0.1`.

On the other hand, Command~11 forwards captured packets to `sample.com` through **ssh(1)**. This example shows a unidirectional ssh tunnel. If a bidirectional ssh tunnel is required, the same setting should be configured on `sample.com`.

5.5.5 Learning switch

Next, we show an example of shell scripting of complicated packet processing using broadcast transfer.

Fig. 6 shows the code of learning switch which has four ports. By using the network scripting environment, a full-function Learning switch can be developed in less than 50 lines. The example is developed using existing shell and UNIX commands. If a network script isn't fast enough, you can use the APIs to write simple C utilities that can then be used in shell scripts.

In particular, computationally intensive functions and common network functions such as Forwarding DataBase (FDB) and routing database should be implemented as a command using in network script.

5.5.6 Existing network tools compatibility

More advanced packet processing needs the functionality of various layers. For instance, the negotiation of ARP packet with network device and IP address setup is a function of both layer 2 and layer 3.

On the other hand, Ethernet Character Device is an input-and-output device of the packet data of a physical Ethernet port, and does not have those upper layer functions. It is necessary to make a network stack for each network application in the construction of a complicated network application.

Tappipe[34] is Ethernet Character Device application which connects the Linux network stack with the device. Ethernet Character Device provides raw packet data. It's easy to connect the OS network stack to the application simply by converting the frame format.

Command 13 show usage of **tappipe**. In the first line, **tappipe** makes a virtual

Ethernet device using physical port o on EtherPIPE. By specifying an EtherPIPE device as stdout and stdin of **tappipe**, the device is created using Linux TAP.

Because pipeo is at the virtual Ethernet device for OS, it can offload the setup of an IP address, and the work of a data link layer called the negotiation of ARP to the existing network stack. Moreover, existing network software such as **ping**, **dhclient**, and **wireshark** can be used with an EtherPIPE port. By using Network scripting and **tappipe** together, we can narrow our focus to scripting our new network functionality.

Command 13: A virtual device for connecting an EtherPIPE and OS network stack

```
$ tappipe pipe0 </dev/ethpipe/0 \
  > /dev/ethpipe/0 &
$ ifconfig pipe0
$ tcpdump -i pipe0
```

5.6 Discussion

This section describes the limitations of the current EtherPIPE design and its implementation, and discuss the extensions.

5.6.1 Interface namespace

Our current device naming rules cannot express multiple network cards. For supporting multiple network cards, each EtherPIPE device may be put in subdirectory of each cards such as `/dev/ethpipe/slot0/0`. Because the control plane of packet processing is complex, it would be handled well by the network stack of OS.

We also will develop virtual network devices for OS network stack under `/dev/ethpipe/`. Further discussion on the EtherPIPE device namespace is required, however, we do not examine it in detail due to the limitation of space.

5.6.2 Configuration of Interfaces

EtherPIPE currently focuses on lightweight scripting of packets over the data link layer. One of the limitations of the current EtherPIPE is that it ignores metadata of physical devices or socket options for TCP/IP. Ignoring the configuration functions, EtherPIPE

can access packets in a simple way. To handle upper layers, some metadata handling scheme is required in EtherPIPE.

To implement such metadata, we can add other devices that have their own purpose for packet processing. The current EtherPIPE raw interface should be kept for performance. And the EtherPIPE shell interface may need to improve the ASCII format for usability on shell scripting even if it needs to pay a performance penalty. If it needs scalability, a device should be developed to set socket like options or store dynamic parameters in data format.

5.7 Summary

Shell scripting is a powerful approach to manipulating files, however, it has not supported network processing. The EtherPIPE device allows shell scripting to deal with network devices and network I/O in the same manner as file devices and file I/O. Through the development of the EtherPIPE, we have shown that many packet processing operations can be described by chains of standard UNIX commands with standard input/output and pipe.

EtherPIPE is a low-layer network device yet, its data format is simple and easy to handle in commands and scripting languages. Therefore, Pktdev can be used not only for simple network scripting but also for more complex packet processing using scripting languages. We believe that EtherPIPE is suitable for SDN where simple packet manipulations are often required.

Figure 5.9: **A learning switch script**

```
#!/bin/bash
#
# usage: learning_switch.sh </dev/ethpipe/0

MY_PORT="0"
TEMP_DIR="/tmp/"

while true
do
  read FRAME

  DMAC=${FRAME:0:12}
  SMAC=${FRAME:13:12}

  # regist SMAC LEARNING TABLE
  if [ ! -f $TEMP_DIR$SMAC ]; then
    if [ $((0x$SMAC & 0x010000000000)) -eq 0 ]; then
      echo $MY_PORT >$TEMP_DIR$SMAC
    fi
  fi

  # search port number by DMAC
  if [ -f $TEMP_DIR$DMAC ]; then
    exec 3< $TEMP_DIR$DMAC
    read PORT 0<&3
    exec 3<&-
    echo $FRAME >/dev/ethpipe/$PORT
  else
    # flooding
    if [ ! $MY_PORT == "0" ]; then
      echo $FRAME >/dev/ethpipe/0
    fi
    if [ ! $MY_PORT == "1" ]; then
      echo $FRAME >/dev/ethpipe/1
    fi
    if [ ! $MY_PORT == "2" ]; then
      echo $FRAME >/dev/ethpipe/2
    fi
    if [ ! $MY_PORT == "3" ]; then
      echo $FRAME >/dev/ethpipe/3
    fi
  fi
done
```

Chapter 6

Evaluation

This chapter describes the evaluation of our two proposed architecture for shell script-based packet processing and hardware assisted timing control.

Section 6.1 studies throughputs of shell script based packet processing using some UNIX commands. And section 6.2 discusses some EtherPIPE applications and its timing accuracy of packet transmission and receive timing.

6.1 Ethernet Character Device

6.1.1 Potential throughput of shell scripting-based packet processing

We evaluate the performance of general network scripting with a dummy driver. There are two major factors. One is memory access throughput because entire packets are passed between UNIX commands through standard I/O. The other is character-based processing (e.g., string matching) used in UNIX commands. We have developed a dummy device driver for EtherPIPE. When reading from the device, the driver returns a dummy shortest (64 byte) Ethernet frame data pre-populated in the device driver. When writing to the driver, the driver simply copies the data into a buffer in the driver. The driver does not take it into account the timing constraints of the Ethernet specification (e.g., Inter-frame gap). The measurements were performed on a PC with an

Table 6.1: **Measuring Shell command-based packet proceedings using dummy driver**

dummy driver (frame size: 64B)	throughput (MB/s)	throughput / 1GE line-rate
capture[1]	1,097	11.52
MAC address filtering (one rule)[2]	833	8.75
MAC address filtering (five rules)[3]	184	1.93
decap ethernet header[4]	8	0.08

[1] cat /dev/ethpipe/0 > dump

[2] grep "^002222222222" /dev/ethpipe/0 > dump

[3] grep "^001111111111|^002222222222 ... ^005555555555" /dev/ethpipe/0 > dump

[4] cut -d' ' -f4- /dev/ethpipe/0 > dump

1GE line-rate (excluded Ethernet preamble and Interframe Gap): 95.24 MB/s

CPU: Intel Core i5 760 2.80 GHz

Intel Core i5 760 running at 2.8 GHz and a ramdisk (tmpfs).

Table 6.1 shows the throughput of typical applications of network scripting using the dummy driver. The results show that simple packet capturing by **cat(1)** achieves more than 10 Gbps, but header rewriting using **grep(1)** and **cut(1)** is much slower.

Modern PCs have enough memory bandwidth (e.g., 10.6 GB/s for DDR3-1333) so that memory copy is not a bottleneck on simple network scripting. Moreover, we can take advantage of multi-core processors and their shared cache when using piped commands.

On the other hand, string matching used in header rewriting requires us to process data byte-by-byte. Many UNIX commands are line-oriented and need to check every byte in search for newline characters. Also, a string match stops when a match is found, but needs to search to the end of a packet when no match is found. It becomes even worse when regular expressions require backtracking. Therefore, the performance of network scripting is heavily influenced by the string matching rules used in a com-

mand.

To further improve the performance, one possible way is to provide a special command to extract specific header fields and apply commands only to the extracted field. For example, to apply **grep(1)** to only the Ethernet headers in packets, it would look something like "**epcmd –extract etherheader –command 'grep PATTERN'** ". Another way is to keep the command syntax but add hardware-based offloading functions to make use of GPU, FPGA or other parallel processing methods.

6.1.2 PPS of Ethernet Character Device on Linux with a comodity NIC

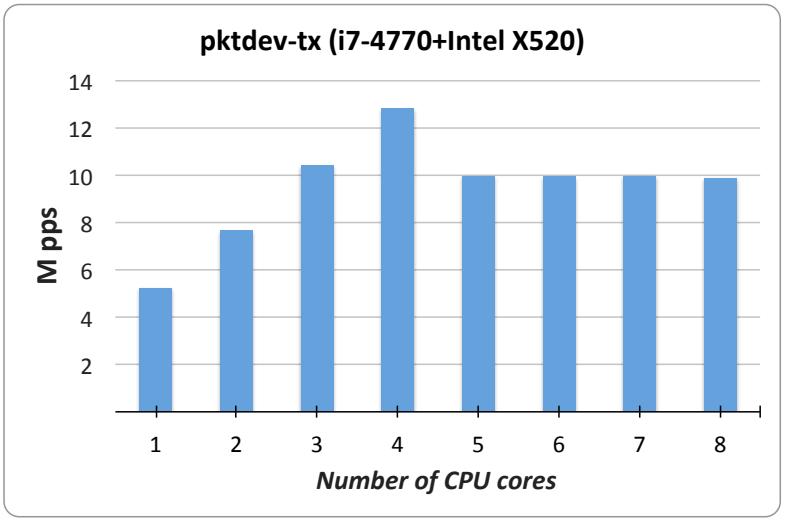

Figure 6.1: **TX Performance of Ethernet Character device using Intel 82599 10G NIC**

Fig 6.1 shows the performance of transmission by binary format of Ethernet Character Device using Intel 82599 10G NIC. We developed a yet another Ethernet Character Device implementation for Linux native device driver by Intel 10G Ethernet NIC to measure actual 10Gbps environment. In the result, performance of Ethernet Character Device is the 12.6 Mpps using 4 CPU core. The reason of performance decreasing on CPU 5 8 core is this PC use hyper-threading and the number of physical CPU is 4

cores. Thus, the limitation of the Linux with Intel NIC device driver is about 12 or 13 Mpps[3]. So this architecture achieved the almost 10G line rate.

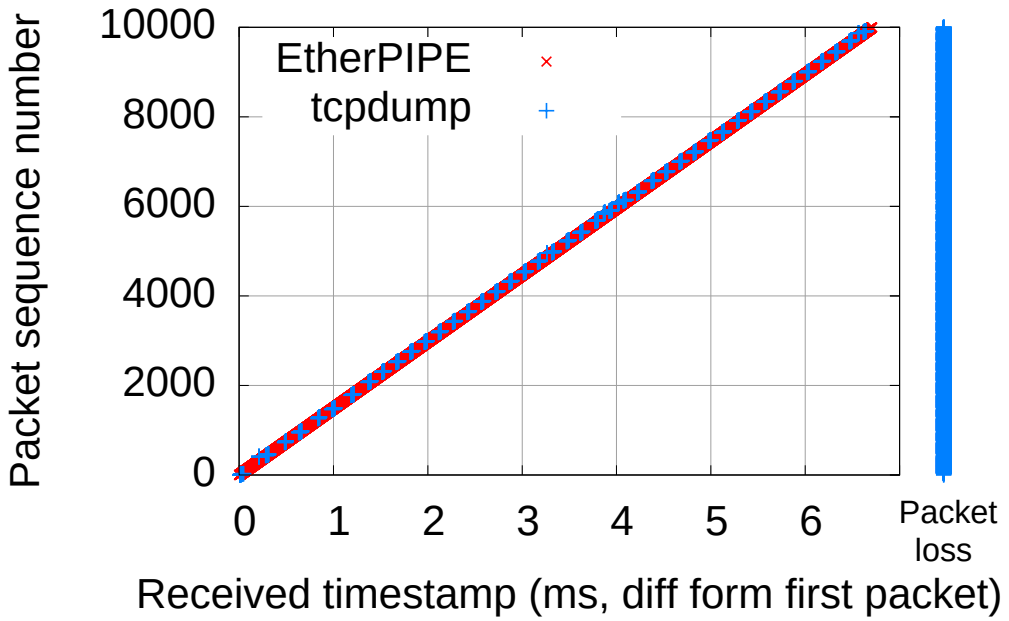

Figure 6.2: **Sending 64 byte packets by pktgen and received by 'cat /dev/ethpipe/0' and tcpdump**

6.2 Performance of EtherPIPE NIC

6.2.1 Received timestamp

Ethernet Character Device can be used to rapidly build simple network tools such as packet capturing and generating tools. In particular, the performance of "cat /dev/ethpipe/0" and "cat ./pkt > /dev/ethpipe/0" shows primitive throughput, and is important in network scripting. Therefore, we compared the throughput of our FPGA-based implementation with that of **tcpdump(1)**, by "cat /dev/ethpipe/0".

This evaluation has two purposes. One is to confirm the data transfer performance of our hardware design and implementation. The other is to show the accuracy of the hardware timestamp in our FPGA implementation.

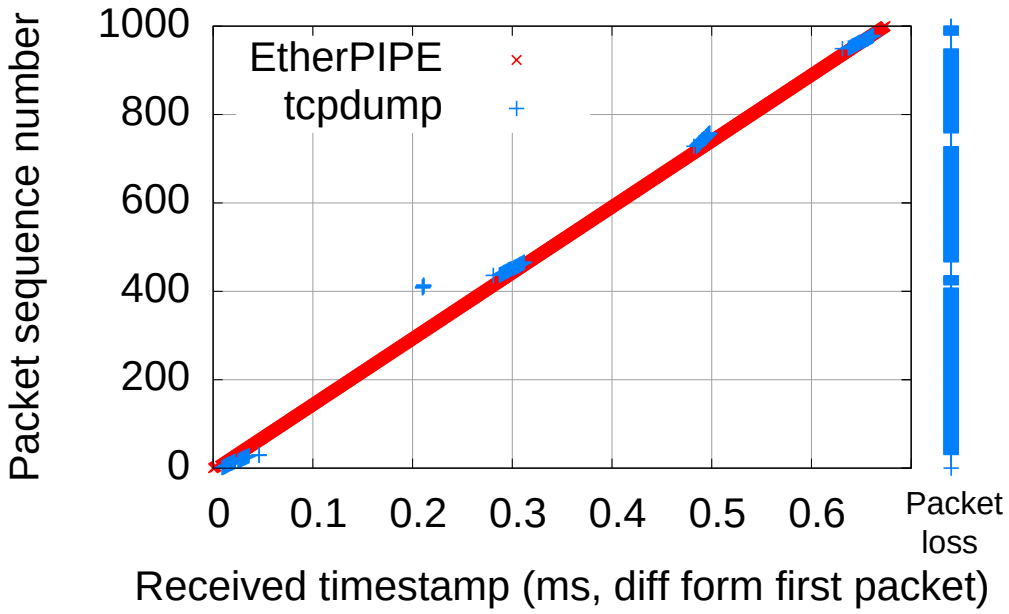

Figure 6.3: **Sending 64 byte packets by pktgen and received by 'cat /dev/ethpipe/o' and tcpdump (First 1000 packets)**

We compare the performance of the FPGA-based EtherPIPE implementation with that of **tcpdump(1)** on a commodity NIC. The capturing machine is equipped with Intel Core i3-3220 and Intel 82579LM Gigabit Ethernet NIC.

The sender uses the Linux pktgen tool to send 10,000 packets with 64 or 1518 byte-long frame sizes. The sender transmits packets at the line-rate so that packets of the same size should arrive with the corresponding constant interval at the receiver.

Figure 6.2, 6.3 shows the evolution of packet sequence number with received timestamp for 64-byte-long packets. The received timestamp on the X-axis shows the offsets from the timestamp of the first packets, and lost packets are shown at the right end. The left figure shows the entire 10,000 packets, whereas the right figure shows the magnified view for the first 1,000 packets.

As can be observed in the figures, Ethernet Character Device successfully received all packets, while **tcpdump(1)** received only 14.1% of packets and failed to receive 85.9% of packets. The device implementation works at 1Gbps line-rate even with 64-byte-long packets.

Figure 6.4 shows the performance for 1518-byte-long packets; only the first 200

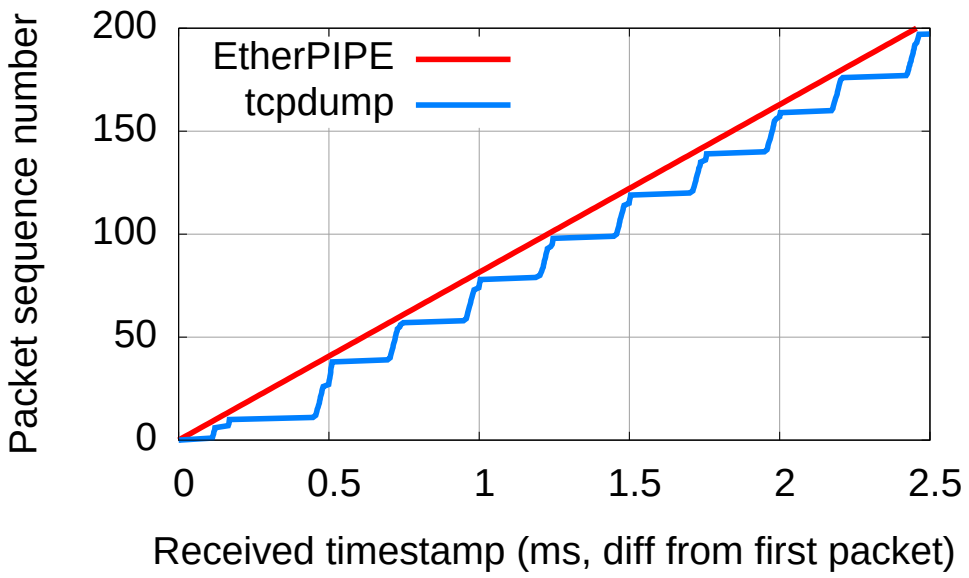

Figure 6.4: **The result of 200 packets which sending 1518 byte packets by pktgen and received by 'cat /dev/ethpipe/0' and tcpdump**

packets are shown in the figure. This time, **tcpdump(1)** was able to receive all the packets without any packet loss. However, the evolution of the timestamp is very bursty for **tcpdump(1)** because timestamps are taken by software when multiple packets are processed at a time. [1] On the other hand, the timestamp of Ethernet Character Device evolves linearly, reflecting the theoretical packet arrival interval at 1Gbps line-rate.

These results show that our Ethernet Character Device implementation can work at 1Gbps line-rate even with 64-byte-long shortest size packets, as well as the accuracy and benefit of hardware-based timestamp.

6.3 EtherPIPE applications

In this section, we discuss some EtherPIPE applications and its timing accuracy of packet transmission and receive timing.

[1] The timestamp of tcpdump is shifted to the right for the delay of the first timestamp so as to not exceed the theoretical value of timestamp.

Figure 6.5: **Scheduled transmission (10 ms)**

```
#!/bin/bash
# usage: ./oneway.sh > /dev/pktdev/1

delay=$(( 10000000 / 8 )) # 10 ms

pkt="$(cat ./ping_echo.pkt)"
ts=0

printf "1100000000000000 $pkt\n" $ts
for (( j=0; j<100; j++ ))
do
      ts=$(( $ts + $delay ))
      printf "01%014X $pkt\n" $ts
done

echo "Done" > /dev/stderr
```

The evaluation environment consists of two Host PCs with our FPGA-based NICs. We developed *oneway.sh*, *ping.sh* and *delay.sh* for measuring the accuracy of EtherPIPE packet timing control architecture. The each shell code only use bash functions and source code is 1050 lines.

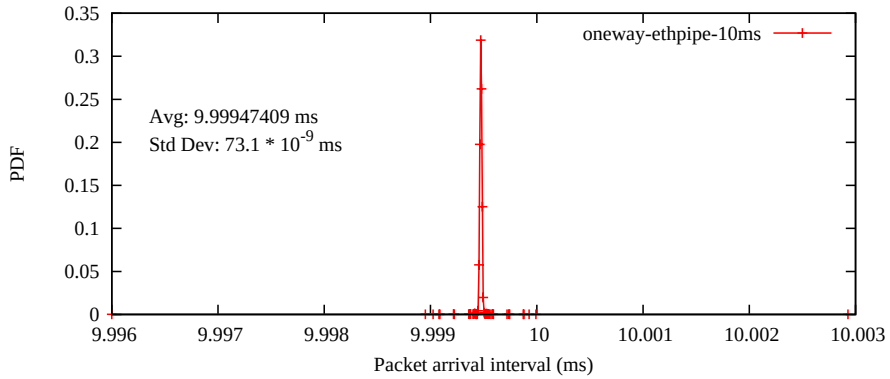

Figure 6.6: **The result of arrival time interval by oneway.sh**

Figure 6.7: **Delay emulation (1 sec)**

```
#!/bin/bash
# usage: ./sec.sh < /dev/pktdev/0 > /dev/pktdev/1

delay=$(( 1000000000 / 8 )) # 1 sec

while true
do
      read pkt # recv

      if [[ $pkt =~ ^[0-9A-F]{16} ]]; then
            recv_ts=${pkt:0:16}
            frame=${pkt:16}

            printf "%016X$frame\n" $(( 16#$recv_ts + 10#
               $delay )) # send
#  echo "received: ts=$recv_ts" > /dev/stderr
      fi
done
```

6.3.1 Scheduled transmission

First, we evaluate the timing accuracy of scheduled transmission by using *oneway.sh*. *oneway.sh* [Fig 6.5] send packets with 10 ms interval. And the receiver PC capture the packets by *cat /dev/ethpipe/1* one-line command.

Fig 6.6 shows the result of packet arrival time interval on receiver PC. In the result, the average interval is about 9.9947409 ms and standard devision is 73.1 ns.

6.3.2 Ping and delay emulation

Next, we evaluate the timing accuracy of the delay emulation script and RTT script. The *ping.sh* [Fig 6.11] script is able to measure RTT with 8 nanosecond accuracy and it can measure the PHY chip processing delay and a length of cables. And the *delay.sh* [FIg 6.7] script can emulate one second delay over the total latency of packet processing on the software.

Fig 6.8 shows the measurement environment of *ping.sh* and *delay.sh*.

And we compare the EtherPIPE delay accuracy with Linux kernel delay emulation function. Fig 6.9 and 6.10 show the result of RTT value of EtherPIPE and Linux one

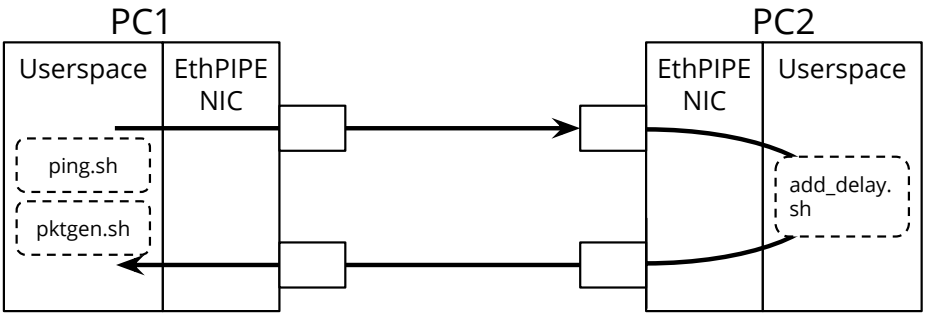

Figure 6.8: **EtherPIPE evaluation setup**

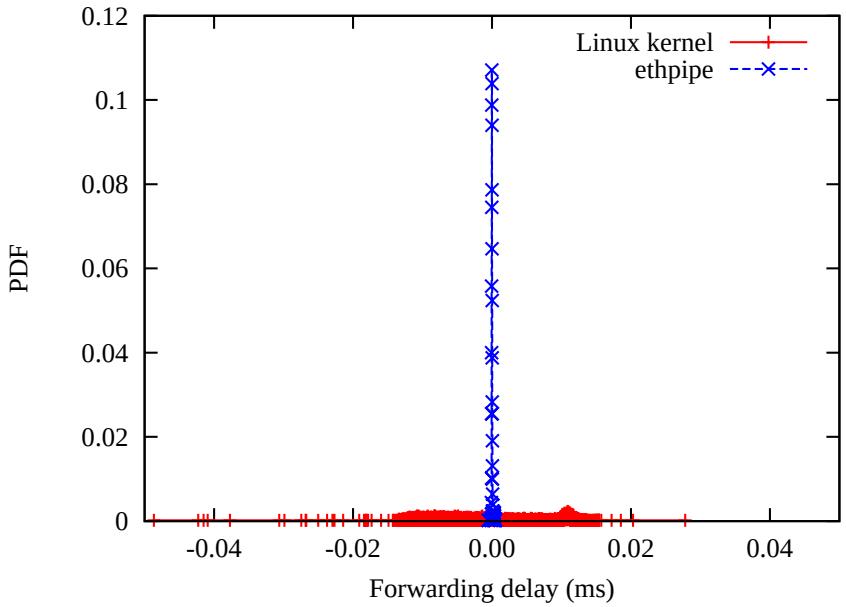

Figure 6.9: **1 second delay emulation by linux kernel and EtherPIPE**

second delay emulation. From Linux delay emulation, average delay is 997.832 ms and standard devision is 8640.7 ns. And EtherPIPE result, average delay is 1000.053 ms and standard devision is 58.25 ns. Using our EtherPIPE implementation, the timing accuracy of packet processing was improved about 148 times than software delay emulation by Linux kernel.

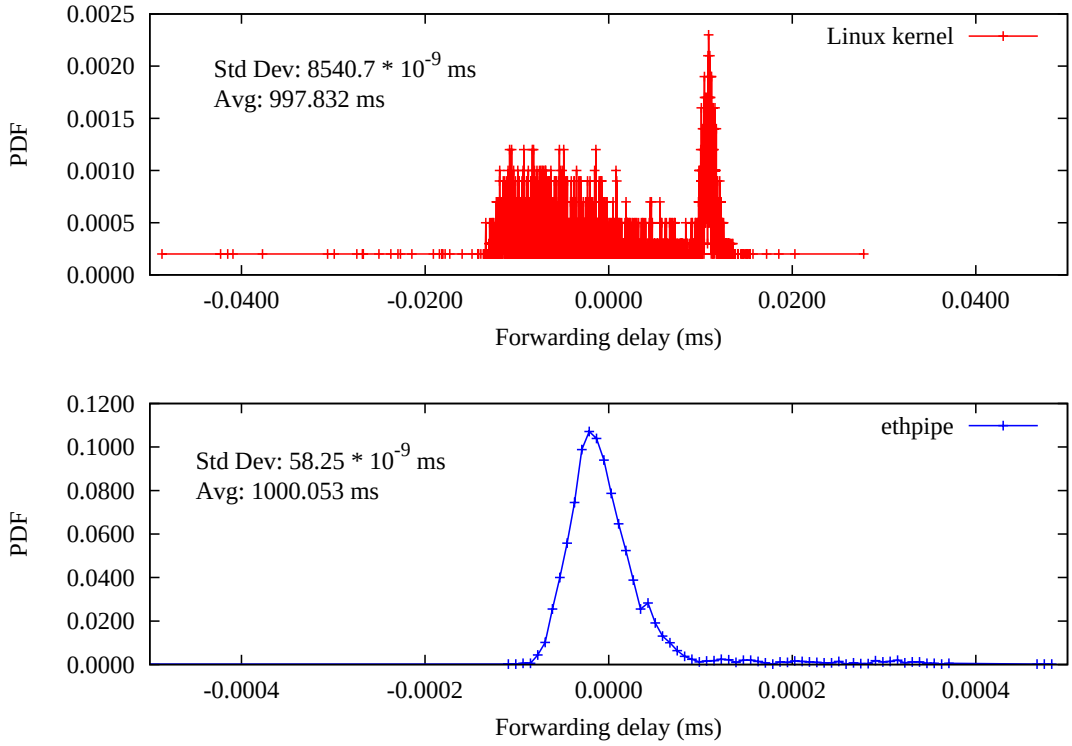

Figure 6.10: **1 second delay emulation by linux kernel and EtherPIPE**

Figure 6.11: **ping.sh**

```
#!/bin/bash
# usage: ./ping.sh < /dev/pktdev/0 > /dev/pktdev/0
#

while true
do

      ...
      build a ping request packet
      ...

      # send PING request
      printf "ping␣from␣%s.%s.%s.%s␣to␣%s.%s.%s.%s\n" ${
         SRC_IP_ADDR} ${DST_IP_ADDR}
      echo $PING_REQ

      while true
      do
            read recv

            if [[ $recv =~ "${PING_ID}␣${PING_NO_TMP:0:2}
               ␣${PING_NO_TMP:2:4}" ]]; then
                  echo "pong" > /dev/stderr
                  TX_TIME=‘cat /sys/kernel/pktdev/
                     local_time1‘
                  RX_TIME=${recv:0:16}
                  printf "RTT:␣%d␣ns\n" $(( (0x${RX_TIME}
                      - 0x${TX_TIME}) * 8 )) > /dev/stderr
                  break
            fi
      done

      sleep 1

done
```

Chapter 7

Case Study

High-precision packet timing control has not only performance impact for local network benchmarking, Internet measurement can be expected to improve the measurement accuracy. This chapter describes the inferring methods and analysis of the Regional AS topology by existing Internet measurement tools[27, 54]. The proposed method enables to infer the continent-level AS topologies using traceroute data measured from multiple vantage points. But the method can't scale with the detail of continent-level AS topology analysis because RTT accuracy isn't enough for inferring the country and city-level AS topology. EtherPIPE is expected to be utilized in the analysis of more detail of Internet structure.

7.1 On inferring Regional AS topologies

The section describe the technique to compare Asian Internet structure in each years and visualize the development of the Asian Internet.

Internet structure is more large and complex in accordance with the development of international traffic. In the background of the growth of international traffic, there is economic growth that information infrastructure strategy in each country. In recent years, cloud providers called hyper-giants such as Google and Akamai tend to operate their own networks. These ASes expands the region of world wide directly, and the traffic exchange closed in the each region.

In general, the analysis of changes in large Internet architecture, AS topology anal-

ysis is an effective means. However, AS topology is a representation of a logical connection relationship between AS, is independent of the physical geographic information. Only current analysis techniques, such as connection bases such as regional strategy and regional IX by country level, it is difficult to closely observe the changes in the macro Internet structure related to the region. Further, in order to physically extensive disorders such as cutting of the submarine cable is to analyze the impact of the Internet, but require analysis of AS topology data in consideration of the geographical information, current analysis techniques is established is not without analysis of each incident is made based on operational experience.

The purpose of this study is to establish a method of analysis Internet structure by local perspective. The method enables a Internet topology analysis method of area in which the network measurement technique based on, especially to perform the analysis of regional development of Asia. The approach that we focus on the operation method of the network to infer the AS connection between the bases location originally is intended to correspond to AS topology data that is independent of geographical information. We believed to affect the connections between AS logical connections from router configuration, data center location and the arrangement of the submarine cable. This analysis technique allows to take into account physical limitations on these management, BGP router position used for AS connections to infer the position of the AS connection. In addition to what AS and AS are connected, where it is possible to focus on or between the AS is connected, the connection information between the AS originally represent only the logical connection relationship, physical such countries and continents and clarify the relationship of the Na position. By this method, only the existing approach has enabled analysis of AS topology considering spatial analysis is difficult.

The proposed method has the same data as the existing AS topology measurements collected by the network measurement technique, BGP data, traceroute data, based on the DNS data. These measurement data, it is recorded by the research organization of network measurement, respectively, is widely exposed to the research community. In this section, we used these past measurement data, and review of the measurement data of 2001-2010, it was actually compare the AS topology in Asia in 2004 and 2010. As a result, the main AS and its connection relationship in Asia, even, was difficult in the existing global AS topology analysis, is Asia outside of AS organization active in Asia became possible grasp. Moreover, the proposed method makes use of the traceroute

data measured from the multipoint. Therefore, this approach cannot parse the areas where there is no measurement data.

7.2 IP Geo-location techniques for inferring location of AS links

In general, when the customer ISP bought IP transit from the provider ISP, they set up town routers in the data center of the provider ISP. Or to extend the access line to the own data center, BGP connections between routers are placed in the same data center. BGP border router IP address pair this topology at the IP level located AS boundary becomes physically be located in the same data center.

Figure 7.1 shows the inferred techniques AS connection between sites where the proposed method. This method collects the inter-AS connection data using the traceroute. Furthermore, this proposed method assumes the IP address of the BGP router is installed IP address pairs AS boundaries found from traceroute data (link AC in the figure) in the same data center. The IP address pair of AS boundary found by the traceroute is referred to as the inter-AS connection.

7.2.1 DNS-based method

There is a case in which AS give a hostname to own router with the own organization which can be inferred geographic information the router location. For example, *xe-1-3-0.r21.tokyjpo1.jp.bb.gin.ntt.net* is a router of the NTT Communications (AS2914), and it has *tokyjpo1* word in hostname, and it can guess the router that exist in Tokyo, Japan.

Scriptroute project[44] of Washington University provides to reverse lookup the IP address of the router in the DNS, have published undns tool to infer the geographic information from the DNS name[53]. undns, the pattern of the host name of the naming convention for the AS 271 at the time in 2008 has been registered. In addition, Washington University iPlane project[52] build the set of own name database for undns.

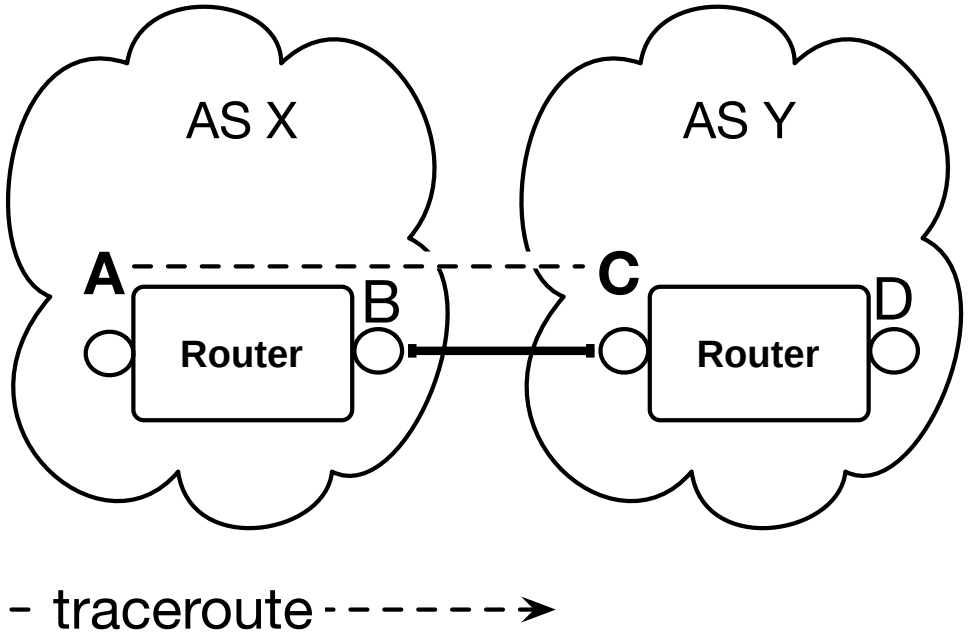

Figure 7.1: **The link AC that was discovered in traceroute is called AS link. This method guesses the pair of IP addresses of AS link exist in the same place**

7.2.2 RTT-based method

The proposed method utilizes the data that was traceroute from physically installed has been measured based on wide area for the entire Internet. Therefore, to estimate the geographic information of AS connections using the IP geolocation method using the RTT value by Gueye et al[20]. This method, previously investigated the geographical information of traceroute measurement bases, to compute the reachable physical range from the measurement base for traceroute measurement results RTT. Furthermore, by utilizing the measurement results from a plurality of measurement locations, to determine the physical existence possible. For certain IP address, and guess the range reached by RTT values from all emerging traceroute base, and guess success only if the location where the ranges overlap was a single continent.

Conversion of RTT values and distance according to the method, the propagation velocity of the packets in the optical fiber to determine the range utilizing RTT 1ms to be approximately 0.6 times the speed of light as 100km.

Figure 7.2 represents a black circle measurement bases executes traceroute, reach-

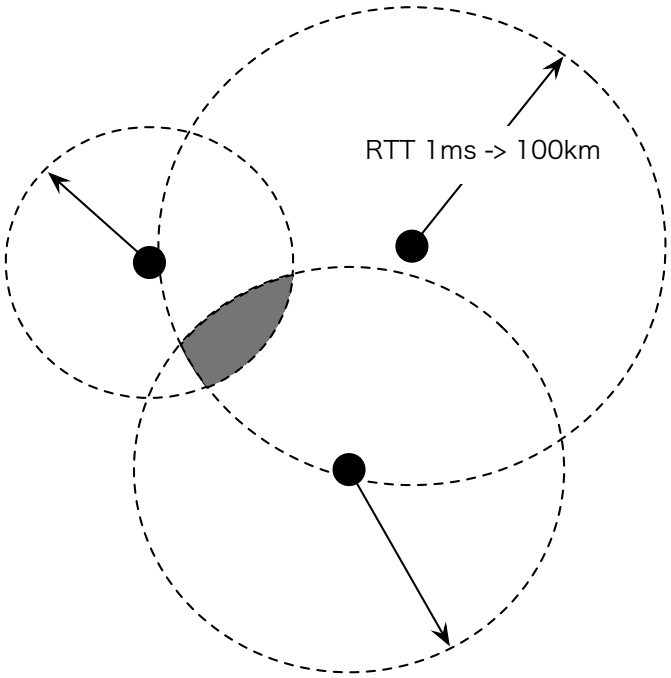

Figure 7.2: **Inferred IP geolocation technique of BGP routers using traceroute data measured from the multipoint**

able range circles with dotted lines based on the RTT value, gray range of IP addresses that narrowed specifically geographically. By using the traceroute data measured from multipoint, to identify a range of existence of the IP address of the AS connection.

7.2.3 Exclusion techniques for exception data

In Section 7.2, General be to focus on inter-AS connection method, it was examined guess how the AS connection between locations. However, the connection mode between the AS, the addition to being present many, might be built in a distributed IX or a wide area L2 on the connection, that case, in this guess method guess incorrectly AS connections location may.

Therefore, in this proposed method, by the use of multiple IP geolocation, and performs elimination of the exception data. First, to infer the geographic information by the procedure described in Section 7.2.1 and Section 7.2.2for the IP address pairs of all the inter-AS connection. At that time, if the IP address pair showed a different

location, that AS connections are assumed not to exist in the same data center, and to eliminate from the data used in the analysis technique described in Section 7.3.

7.3 Techniques for Inferring Regional AS-level Topologies

This section describes the AS topology analysis method considering the geographical information. The proposed method combines the AS topology analysis method using traceroute data, the guess method of inter-AS connection bases mentioned in Section 7.2.

Figure 7.3 describes the proposed method. (1) First, it extracts the inter-AS connection using a traceroute data measured from the multipoint wide area, it is assumed that the IP address pair (dashed lines in the figure) are in the same location. (2) Next, using a traceroute data measured from the multipoint wide area, and any IP address of the extracted AS connection between the key, will continue to search for other inter-AS connection data. Thus, geographic information is unknown I will clustering AS connections that exist in the same data center. Figure AS connections expressed by a dotted line, because they share the same IP address, the IP address of each of the AS connection between the inferred to exist in the same location. (3) Finally, to estimate the specific geographic information of the cluster that was estimated to be present in the same location. In this method, by IP geolocation technique described in Section 7.2.1 and Section 7.2.2, to infer the geographic information from the IP address of the BGP router. So, if the geographical information of any IP address in the cluster can be identified, I will be the IP address of the geographic information of the entire cluster.

7.4 Inferring the regional AS topologies

This section shows the result of continent-level AS topologies by using the proposed method in section 7.3. The continent means Africa, Asia, Europe, Oceania, North America and South America.

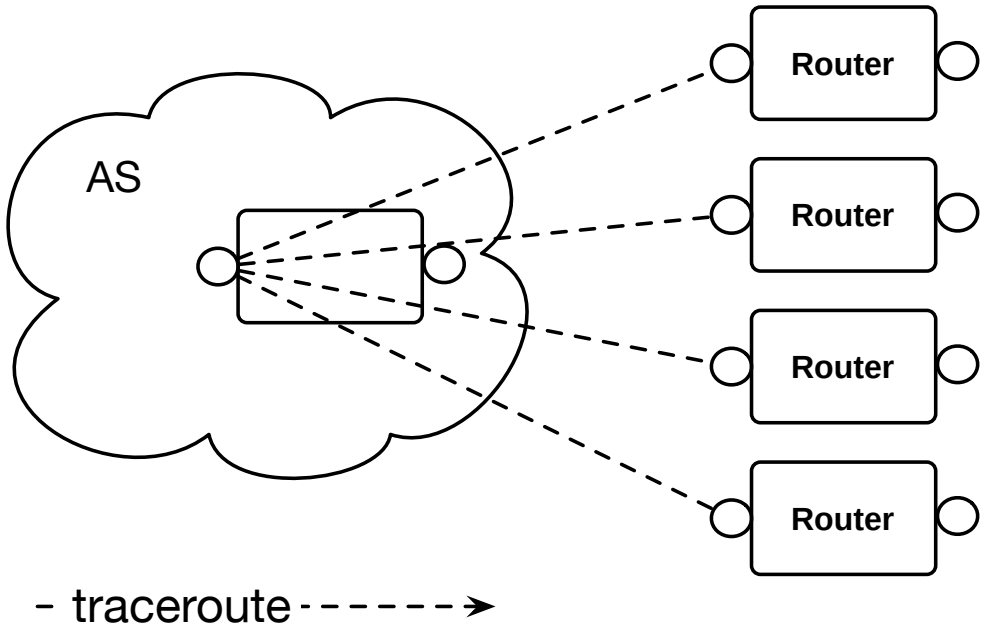

Figure 7.3: **A clustering method IP address of the BGP routers using the location of AS links**

7.4.1 Topology data

This section shows the data used to evaluate the proposed method. The method use the traceroute data from 2001 to 2010, BGP, and DNS data for illustrating the regional AS topology analysis.

Table 7.1 shows data size of traceroute, BGP and DNS data. The proposed method is based on the analysis method of the AS topology with existing traceroute data. traceroute data, I was using the Skitter[14] and Ark[12] data CAIDA project is collecting. In addition, in order to extract the inter-AS connection from traceroute data, you are using the routing table of BGP full route of Routeviews[51] and RIP NCC. Size is the number of attempts traceroute table in traceroute data, size is the number of paths of BGP data, size of DNS data represents reverse lookup the IP address number, respectively.

Skitter and Ark measures the traceroute data of world wide during about three days. Measurement technique is to divide the advertisement prefix in the path of /24, executes a traceroute to any IP addresses in each prefix object. The traceroute data is

Table 7.1: **The data size and measurement years of traceroute data, BGP data and DNS data DNS data**

Year	traceroute data			BGP data			DNS data		
	src	date	size	src	date	size	src	date	size
2001	Skitter	Sep/1-15	59M	Routeviews, RIPE	Sep/1	114K	ISC	-	-
2002	Skitter	Sep/1-15	81M	Routeviews, RIPE	Sep/1	119K	ISC	Q3	238M
2003	Skitter	Sep/1-15	98M	Routeviews, RIPE	Sep/1	136K	ISC	-	-
2004	Skitter	Sep/1-15	146M	Routeviews, RIPE	Sep/1	157K	ISC	Q3	357M
2005	Skitter	Sep/1-15	136M	Routeviews, RIPE	Sep/1	184K	ISC	Q3	419M
2006	Skitter	Sep/1-15	125M	Routeviews, RIPE	Sep/1	210K	ISC	Q3	511M
2007	Skitter	Sep/1-15	96M	Routeviews, RIPE	Sep/1	238K	ISC	Q3	576M
2008	Ark	Sep/1-15	94M	Routeviews, RIPE	Sep/1	283K	ISC	Q3	663M
2009	Ark	Sep/1-15	125M	Routeviews, RIPE	Sep/1	316K	ISC	Q3	791M
2010	Ark	Sep/1-15	154M	Routeviews, RIPE	Sep/1	349K	ISC	Q3	857M

the five cycles of the data at every 1st to 15th September. BGP data, using the data of every year September 1 that Routeviews and RIPE NCC has been published. In addition, as a result of the scrutiny of the data, the traceroute data from 2001 to 2003, RTT value of traceroute has not been archived.

Next, in order of IP geolocation by DNS name mentioned in Section 7.2.1, the method uses the DNS data of the entire IPv4 space. DNS data is collected by the Domain Survey project of ISC[25]. ISC of Domain Survey has measured all IPv4 space on a quarterly basis, this time I was using the data of the DNS name of the Q3. The archive data in 2001/Q3 is broken and can't unarchived.

Figure 7.4 shows the size of BGP data and traceroute data used in this analysis. The white bar in graph shows the total number of AS that has been observed in the traceroute data, and the shaded bar shows all AS number observed from the BGP data, and the line graph represents the traceroute measurement base number of CAIDA Skitter and Ark. AS number that could be observed by the traceroute is decreased from 2005 to 2007 in proportion to the decrease in measurement locations. This relies on the traceroute by the measurement environment CAIDA is reduced from 2004 to 2007. Since

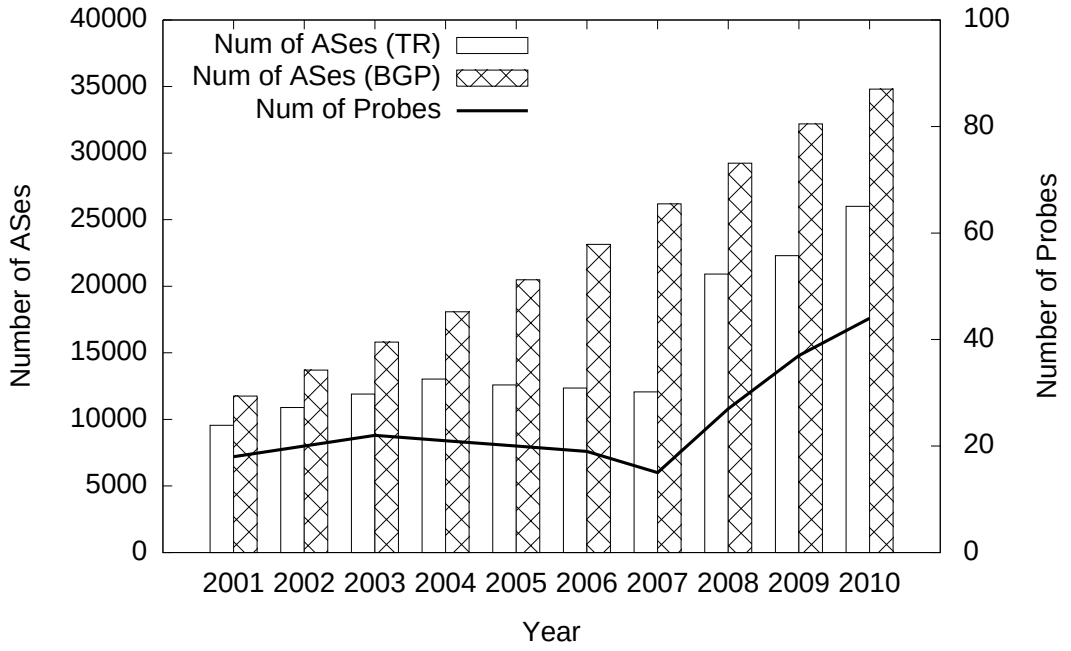

Figure 7.4: **Data size of traceroute and BGP information**

2008, the measurement environment and techniques it is possible to increase the number of sites simultaneously measuring the change in Ark from Skitter, all AS numbers in the measured traceroute data is increasing.

Finally, the organization information such as addresses and AS name of each AS, the method uses RIR data from AfriNIC, APNIC, ARIN, LACNIC, RIPE NCC. In addition, the organization name of AS is from the WHOIS information[1].

7.4.2 Number of the landmarks of geolocation

Next, Table 7.2 shows the analysis results using IP geolocation technique described in Section 7.2.1 and Section 7.2.2. The values in the table shows the number of IP addresses that geographic information can be inferred by the IP geolocation methods from the IP addresses of all inter-AS connection data.

DNS is the data from described in the section 7.2.1, *RTT* technique described in Section 7.2.2, *miss* is the number of IP addresses showed different continents in the

DNS and RTT, *uniq* is at least one IP total number of addresses that you can guess, $uniq/TR$ shows the percentage of uniq for the IP address number that is included in all inter-AS connection. In the method using the DNS data, can not be inferred geographical information only been AS only registered in undns, the method using the RTT value, could guess the location of more IP addresses. Moreover, when comparing the IP address of geographic information can be inferred by both methods, the data to infer a different location in the two methods were not present.

In 2004, but was able to guess continent IP address of 70CAIDA Ark data set that was used from 2008 and later, compared to 2004, China and Africa, the measurement bases in regions such as South America has increased. These areas, measurement base number is less than the United States and Europe, even a wide area of the region. The reason the number of guesses of 2010 has decreased, China, to Africa and South America of IP address, I considered speculation of area using RTT value is difficult. However, the proposed method clusters the IP address in the same data center in the key IP address of the AS connection. Landmark data, it is therefore not necessary to guess the IP address of all inter-AS connection data in auxiliary those used for spatial transformation of each cluster.

In the next section I show the clustering size of each continent is the result analysis (AS between the number of connections).

7.4.3 Inferring continent-level AS topologies

Table7.3 shows the number of connections between different AS continent in 2004 to 2010. The traceroute technique is used in this analysis, because it is easy to measure the major ISP or regional IX as appearing in the path to the destination IP address, the US and Europe, easily discover routes in Asia. On the other hand, when viewed from a macroscopic point of view, such as Africa and South America, is difficult path observation area located in the end point of the Internet. In these areas, it is considered necessary to measure to place the measurement locations in that area. Table in Unknown, this is the proposed method represents the inter-AS number of connection data that could not guess geographical information, in parentheses represent the percentage of from between the whole of the AS number of connections.

Table 7.2: **The resulting number of could be inferred data**

Year	DNS	RTT	miss	uniq	$uniq/TR$
2001	30K	-	-	30K	18%
2002	33K	-	-	33K	22%
2003	36K	-	-	36K	19%
2004	36K	98K	0	116K	70%
2005	41K	99K	0	119K	59%
2006	92K	113K	0	138K	58%
2007	37K	80K	0	104K	51%
2008	114K	123K	0	219K	25%
2009	150K	137K	0	265K	24%
2010	177K	206K	0	358K	18%

7.5 Comparison of Asian AS topology of 2004 and 2010

Next, we compared the Asian AS topology during the 2004 and 2010 by using the topology data analyzed by this analysis method. The result is was performed to compare by visualization using the AS Core Map technique of CAIDA, comparison of distribution due to the inter-AS connection number of major Asian countries, a comparison of the leading AS to focus the inter-AS connection number of countries. In this dissertation, the hub AS is referred as a AS connected with international and regional AS links.

7.5.1 Comparison of AS topology using CAIDA AS Core MAP

First, the section analyzed the Asian AS topology by using the AS Core Map technique of CAIDA.

AS Core Map is a visualization technique of AS topology by Skitter project of CAIDA. This method is visualized using the number of AS links of each ASes and organization address. The map plots ASes by mapping the registered geographic loca-

Table 7.3: **comparison of the number of AS links in each continent**

Year	Africa	Asia	Europe	Oceania	NorthAmerica	SouthAmerica	Unknown
2004	661	22K	59K	4K	94K	5K	23K(11%)
2005	268	34K	57K	4K	105K	12K	18K(8%)
2006	0	14K	39K	0	92K	0	26K(15%)
2007	302	22K	56K	6K	99K	13K	21K(10%)
2008	546	75K	206K	10K	323K	34K	112K(15%)
2009	830	79K	258K	10K	395K	42K	158K(17%)
2010	1,327	97K	438K	12K	825K	50K	197K(12%)

tion of the AS in the WHOIS database to the corresponding longitude. A link between two ASes shows that the two ASes are directly connected. The radius of an AS, the distance from the center, is computed based on the out-degree of the AS that is a number of its outgoing links (shown in Equation 7.1) so that ASes with higher out-degree are placed closer to the center. $radius(AS_i)$ is, AS_i of radius, $outdegree(AS_i)$ is AS_i of out-degree, $outdegree(AS_{max})$ is the most out-degree is often AS_{max} means number of the out-degree. By Equation (7.1), and most out-degree are placed close Fig mainly AS often, to determine the placement of each standard the out-degree of each AS. The AS is often out-degree of inter-AS connection is disposed near the center of the figure, AS outdegree is small is positioned near the outer periphery. Out-degree and that it be placed in the center of the circle is large AS, AS that is the Internet core is disposed in the center of the graph connected by a hub AS is emphasized.

$$radius(AS_i) = 1 - \log\left(\frac{outdegree(AS_i)+1}{outdegree(AS_{max})+1}\right) \tag{7.1}$$

Figure 7.5 and 7.6 show the result of the Asian AS topology in 2004 and 2010 using the AS Core Map visualizing of CAIDA. In the figure longitude 70E range from (70 degrees east longitude) of 140E (in the figure upper left) is approximate Asia and Oceania, under in 120W (figure from 60E (in the figure upper right) is Europe and Africa, 60W (west longitude 60 degrees) from 0) AS of North America and South America are

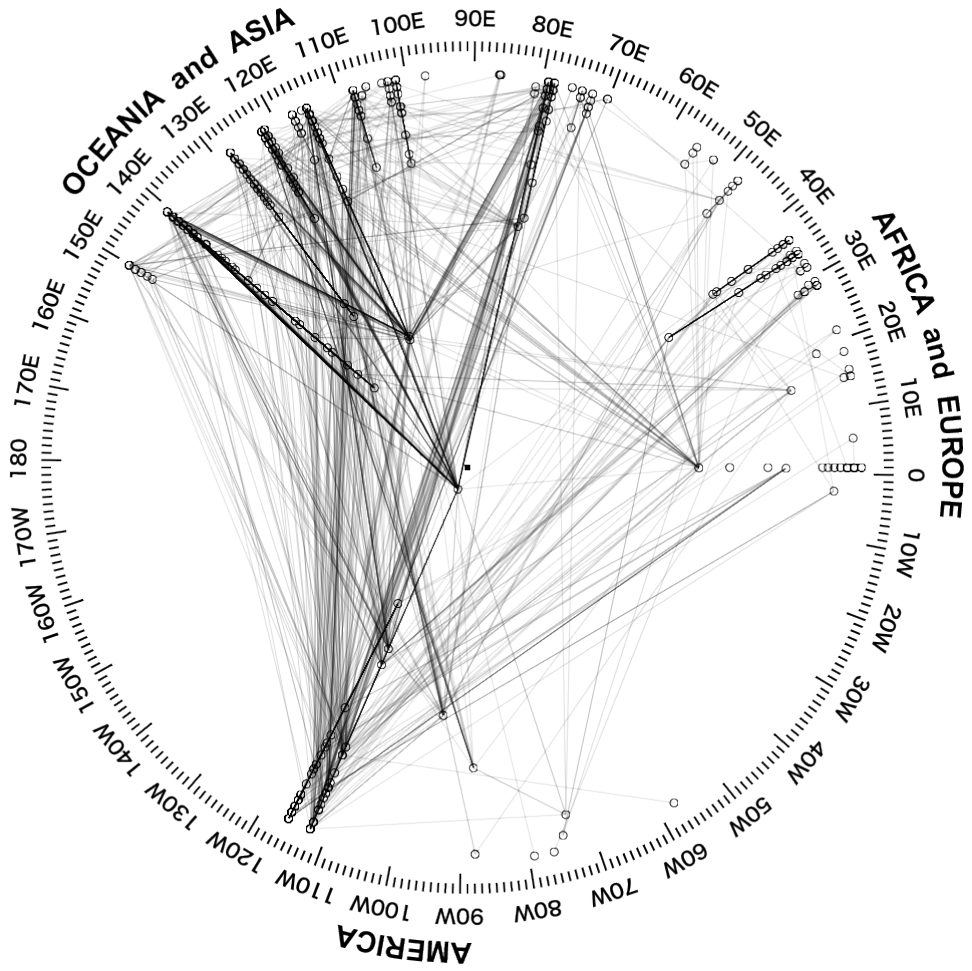

Figure 7.5: **Asian AS topology in 2004**

located in. For example, AS Japanese tissue is placed at an angle of 139 degrees east longitude.

In Asia in 2010 in Figure 7.6, as compared to 2004, confirmed that the upper left of the line overlaps the complex. It has been that the AS Asian countries are tightly connected within Asia and from Asia AS, can also be confirmed connection with US AS located below in FIG. An AS connection between in Asia, not only Asian countries, it has been shown that are many connection with the US AS doing business in Asia.

In Asia AS topology 2004, several Japanese AS is located closer to the center, I can be seen that the whole of Asia AS topology is formed. In contrast, in 2010, AS of each

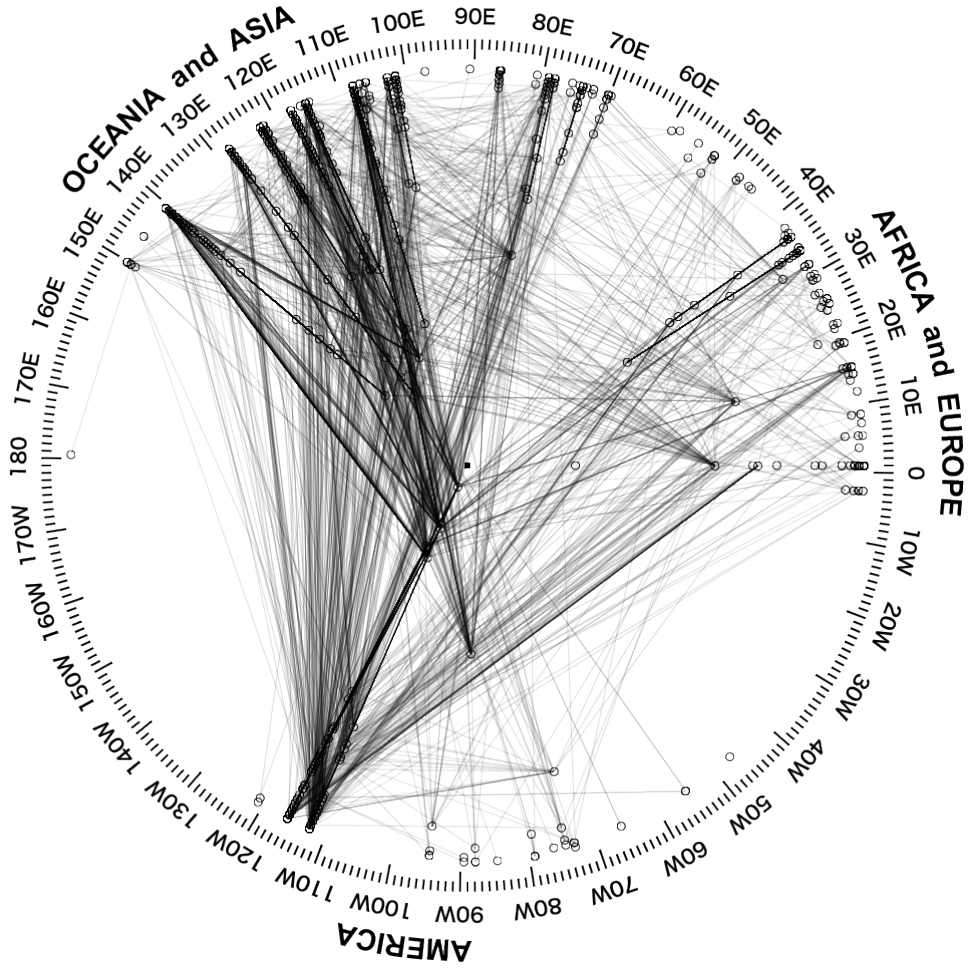

Figure 7.6: **Asian AS topology in 2010**

major countries of Asia can be confirmed. With the growth of Asian Internet, Asia AS to perform the major countries in Thai of AS connection was confirmed appearance that you are experiencing.

7.5.2 Comparison of number of out-degrees on hub ASes in Asia

This section focus on the hub AS of major Asian countries. Table 7.4 shows those listed the out-degree order to 20 organization of inter-AS connections in Asia in 2004 and 2010. In the table, ASN means AS number, AS Name is AS organization name, CC

Table 7.4: **Ranking of Asian ASes of out-degrees of AS links in 2004 and 2010**

	2004				2010			
	ASN	AS Name	CC	degree	ASN	AS Name	CC	degree
1	2914	NTT	US	177	3491	PCCW	US	277
2	2516	KDDI	JP	90	3356	LEVEL3	US	206
3	4637	REACH	HK	79	4766	KIXS-AS-KR	KR	181
4	7527	JPIX	JP	77	3216	SOVAM-AS	RU	177
5	10026	PACNET	HK	77	2914	NTT	US	165
6	3216	SOVAM-AS	RU	72	2516	KDDI	JP	157
7	2497	IIJ	JP	69	1299	TELIANET	EU	157
8	4725	ODN	JP	59	3786	LGDACOM	KR	154
9	4713	OCN	JP	56	10026	PACNET	HK	146
10	3786	LGDACOM	KR	54	9318	HANARO-AS	KR	129
11	703	UUNET	US	50	23947	CEPATNET-AS-ID	ID	117
12	4694	IDC	JP	49	9729	IS-AP	HK	116
13	4766	KIXS-AS-KR	KR	47	2497	IIJ	JP	98
14	1239	SPRINTLINK	US	43	7527	JPIX	JP	91
15	4716	POWEREDCOM	JP	41	18302	SKG_NW-AS-KR	KR	87
16	2907	SINET-AS	JP	39	6453	GLOBEINTERNET	CA	85
17	15412	FLAG-AS	GB	37	9121	TTNET	TR	84
18	9121	TTNET	TR	34	4713	OCN	JP	81
19	7473	SINGTEL-AS-AP	SG	32	4725	ODN	JP	72
20	5511	OPENTRANSIT	FR	31	9304	HUTCHISON-AS-AP	HK	68

notation by country code country information of AS organization, degree represents each the out-degree of inter-AS connection.

Although the 2004 Asian can confirm 8 organization of Japan, is a 5 organization in 2010 against it. In addition, in the ranking of 2010, has increased the AS of Hong Kong (HK) and South Korea (KR), in Asia, Russia AS (RU) was confirmed to be very large. In

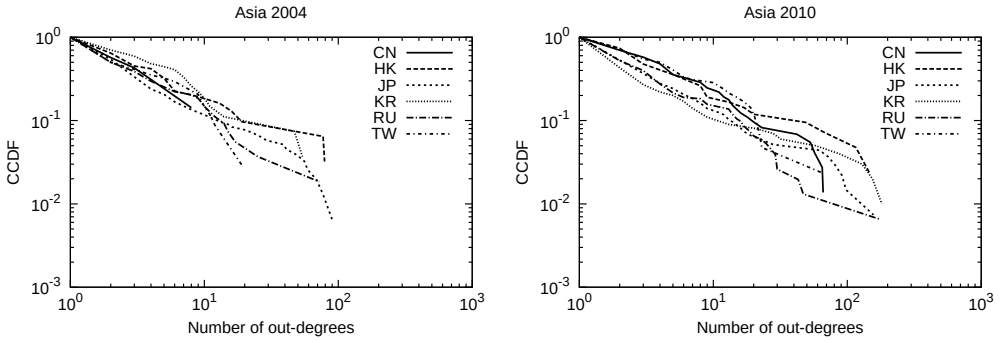

Figure 7.7: **CCDF of node degrees for Asian AS topology in 2004 and 2010**

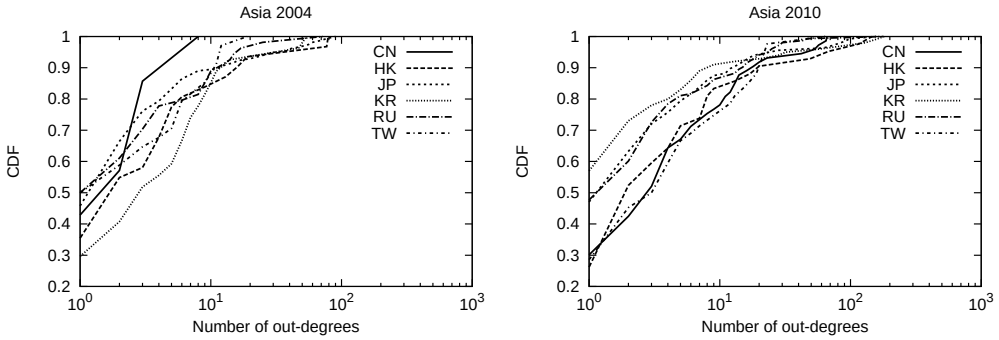

Figure 7.8: **CDF of node degrees for Asian AS topology in 2004 and 2010**

addition, NTT from Japan can be confirmed in the ranking (JP) (AS2914), Korea KIXS-AS-KR (KT, AS4766) of (KR), Turkey TTNET (AS9121) is a ISP in each country of the largest.

7.5.3 Comparison of the distribution of the inter-AS connections

Finally, the section shows the distribution by out-degree of inter-AS number of connections by country of organization in Asia.

Figures 7.5.3 and 7.5.3 in 2004 and CCDF by AS between the number of connections in 2010, as in figure 7.7 and 7.7 represent the CDF by AS between the number of

Table 7.5: **Country average and Top 5 average number of out-degrees in 2004 and 2010**

		CN	HK	JP	KR	RU	TW
2004	mean	2.71	8.77	6.05	7.81	5.13	4.06
	top 5%	8.00	79.00	63.00	54.00	48.50	19.00
2010	mean	8.44	12.71	7.43	8.87	5.92	7.67
	top 5%	62.67	131.00	93.50	121.80	54.29	43.50

connections in 2004 and 2010. Table 7.5 represents the 2004 and the country of average out-degree in Asia of 2010, the average out-degree of out-degree order top 5 % of AS as a hub AS of countries. Countries in the figure is selected from the country emerging AS organized in Table 7.4.

In comparison by CCDF of figures 7.7 and 7.7, Asian countries, especially Taiwan (TW), China (CN), grows to become the hub AS in Asia. In addition, Table 2, Hong Kong, Taiwan, the average degree of China is rising.

Next, in the comparison by CDF in Figure 2 and 3, out-degree are noted AS of about 1 to 10. First, Korea (KR), has been an increase in the proportion of small and medium-sized AS number of Hong Kong (HK). The growth of the information infrastructure in each country is necessary to go into the active participation of these small ISPs. In addition, Table 7.5 shows the growth of Hong Kong and Taiwan presences.

7.6 Summary

This chapter shows the inferring method of regional AS topology by using traceroute, DNS, BGP data from each research groups. The proposed method enables to extract the actual AS links from ambiguous measurement data such as the route by RTT value or traceroute, and visualized from the macro viewpoint of the local Internet by combining them. The Data used in the proposed method are common measurement data gathered and archived by the research organization. The analysis result shows comparing the Asian AS topology in 2004 and 2010, and visualized the growth of Taiwan and Hong Kong AS in Asia. In addition, the result clarifies some regional AS growth such as TTNET in Turkey.

The current method has many limitation which is difficult to visualize areas of the network is developing, such as Africa and South America. In the future, we plan by improvement of adding and methods of data, consider a visualization method of AS topology structure of relatively measurement data is small region.

Moreover, AS rankings are used in the current hub AS analysis contains IX AS number. The IX AS number is necessary to devise such as exclude IX by aggregating as inter-AS connection. Furthermore, in the current analysis, it is aggregated ranking in out-degree of each AS number. ISP companies, including NTT has distinguish the plurality of AS numbers in the operation, it does not consider them in this paper. The related works by Cai et al.[11] proposed a method to analyze aggregate the AS number for each organization. As future work, we considered necessary analysis that takes into account the trend of acquisitions of analysis and ISP of each organization.

Chapter 8

Related Works

8.1 Network device abstractions

The concept of character-based network interfaces is not new. STREAMS[16] employs a modular architecture for implementing I/O between device drivers including network subsystems. Plan 9[39] pushes it further to abstract everything including network as a file, and controls the network stack and services through files. Other systems such as x-kernel[23] and Netgraph[18] provide a framework for building a network stack by connecting protocol modules. The main focus of these systems is to provide an abstraction of network interfaces and protocol stack components.

There exist network interface devices that allow the programmer to access Ethernet frames such as DLPI (Data Link Provider Interface)[38], a STREAMS device driver of SunOS, and the TUN/TAP device driver. They are often used to implement a tunneling or bridging function in user space.

The purpose of the Ethernet Character device is to allow network scripting. To this end, it provides a simple abstraction of Ethernet ports as a character device, and converts Ethernet frames to and from ASCII representation for easy processing by UNIX commands.

8.2 Internet measurement

AS topology has been widely analyzed as basic data of the Internet of actual conditions and the next-generation Internet architecture examination of from the macro perspective. Many AS topology analysis, BGP data, traceroute data can be classified into three types by the data to be used in the WHOIS data[33].

Technique using BGP data is used to analyze the AS topology. BGP is the only routing protocol that is used between AS and it is a path vector type routing method. BGP has AS path attribute which manages all the information of the relay AS number of the destination routing routing table. From the path table of the BGP router with all advertised route called full route, and by referring to the AS path attribute, it is possible to obtain the connection relations between the AS. However, BGP router has a mechanism that collectively multiple paths advertised by aggregating the external route AS. Therefore, only BGP routing table measured from a single AS is grasped all AS connections that exist on the Internet is difficult. Research organizations such as Routeview and RIPE / NCC is externally publish BGP routing table that was observed from the own AS[51, 41]. Also, the Cyclops project UC, have published a variety of by research organizations are analyzed by combining the BGP route information is published, the analysis results by collecting more inter-AS connection data. Cite cyclops.

Then the traceroute data enables to measure the IP address-level Internet topology. In addition, combination of traceroute data and BGP routing table enables to infer the AS topology structure. The AS path attribute of BGP routing table, called the Origin AS, contains the path prefix of advertising the original AS number. Therefore, it was found in traceroute, from the IP address of each router on the path, it is possible to investigate the prefix of the BGP routing table that contains it, associated route IP address and route advertisement original AS number of the traceroute[19]. Check the route advertisement original AS number of the router IP address of the traceroute data, by extracting the connection of AS numbers, it is possible to collect the AS connection between the data using the traceroute.

The CAIDA project based in the University of California, called Ark[12], utilizes the measurement locations which are disposed around the world, it is possible to execute a traceroute from the multipoint, and perform the global topology data collection. In addition, CAIDA has published a graph visualization me a visualization method of AS topology called AS Core Map has proposed[22], was measured AS topology data to this

visualization technique. Other The, DIMES projects and iPlane projects, is collecting AS topology data using a traceroute techniques[47, 32]

Finally, it is possible to acquire the routing policy of the part and the inter-AS connection data by referring to the IRR (Internet Routing Registry) in the WHOIS command. IRR is a pathway database for sharing routing information between network operator. Can investigate inter-AS connection by analyzing the entire AS in the IRR. However, the point that registration of the routing information to the IRR is being performed by any per AS, there is a possibility that erroneous input of the path information generated for the operator's manual input occurs.

Chapter 9

Conclusion

The dissertation introduced EtherPIPE, a new packet scripting architecture with high-precision packet timing control and for flexible development. EtherPIPE aims at rapid and low-cost development of timing-sensitive network applications by means of Ethernet character device that is a packet scripting framework for shell scripting and UNIX commands, and Hardware-assisted Timing-Control NIC hardware.

The Ethernet character device allows shell scripting to deal with network devices and network I/O in the same manner as with file devices and file I/O. The Ethernet Character device can be used for complex packet processing such as packet forwarding and latency emulation using scripting languages. The Hardware-assisted Timing-Control architecture allows receiving a packet with a timestamp in PHY-clock accuracy, and schedules the timing of packet transmission as designated by userspace scripts. The NIC function allows to develop many packet processing applications which require precise packet timing control such as benchmark equipment by software.

The EtherPIPE implemented on a commercial 1G Ethernet FPGA NIC allows to develop a nanosecond ping tool, a delay emulation middlebox and other benchmarking tools. And the evaluation shows that EtherPIPE device achieves 10 Gbps by network scripting.

Overall, EtherPIPE scripting architecture facilitates network hardware verification, research prototyping and Internet measurement. In particular, rEtherPIPE would be useful for recent data center networks equipped with a number of special hardware

acceleration to improve performance of software applications. We believe that EtherPIPE has a potential to be an indispensable debugging tool for developing network applications on mixed environments of software and hardware,

9.1 Future Directions

The timing resolution of packet processing affects the performance of all traffic control, and the EtherPIPE architecture can be expected to improve these performances. For example, to achieve near 100% link utilization of fibers is a hot topic for the network research. It requires all network process would to have hardware-level packet shaping so that EtherPIPE would match the need to solve this future problem.

The future direction of EtherPIPE is the acceleration of functions of traffic control on the existing operating systems. The API of EtherPIPE NIC is only for packet timing control so that it is orthogonal to the existing OS implementations. Although the current EtherPIPE supports only its own special device driver for packet scripting, EtherPIPE NIC can be extended to support the existing framework for the device driver of the OS network stack. Once this is done, EtherPIPE would accelerate packet timing of many traffic control functions and protocols.

The current EtherPIPE prototype is developed on the FPGA NIC with 1G Ethernet ports. Another important future direction of this dissertation is to support 10G or higher bandwidth networking. We will develop 10G EtherPIPE NIC and explore the possibility of EtherPIPE in general usage.

Bibliography

[1] Cidr report. *http://www.cidr-report.org/as2.0/*.

[2] IEEE 1588 Standard for A Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. http://www.nist.gov/el/isd/ieee/ieee1588.cfm.

[3] Linux-netdev - [patch net-next 3/3] packet: introduce packet_qdisc_bypass socket option. *http://www.spinics.net/lists/netdev/msg259978.html*.

[4] traceroute. *http://www.traceroute.org/#source%20code*.

[5] A. Kantee. Environmental Independence: BSD Kernel TCP/IP in User Space. In *Proceedings of AsiaBSDCon'2009*, 2009.

[6] A. Atlas, J. Halpern, S. Hares, D. Ward, and T. Nadeau. An architecture for the interface to the routing system. Internet Draft, IETF, August 2013.

[7] B. Pfaff and J. Pettit and T. Koponen and K. Amidon and M. Casado and S. Shenker. Extending networking into the virtualization layer. In *Proceedings of HotNets-VIII*, Oct. 2009.

[8] P. Biondi. Scapy. http://www.secdev.org/projects/scapy/.

[9] S. Bradner. Benchmarking Terminology for Network Interconnection Devices. RFC 1242 (Informational), July 1991. Updated by RFC 6201.

[10] S. Bradner and J. McQuaid. Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational), Mar. 1999. Updated by RFCs 6201, 6815.

[11] X. Cai, J. Heidemann, B. Krishnamurthy, and W. Willinger. An organization-level view of the Internet and its implications (extended). Technical Report ISI-TR-2009-679, USC/Information Sciences Institute, June 2012.

[12] CAIDA. CAIDA: The Cooperative Association for Internet Data Analysis: CAIDA: Archipelago Measurement Infrastructure. `http://www.caida.org/projects/ark/`.

[13] C. Catlett and G. Foot. Libnet Homepage. `http://libnet.sourceforge.net/`.

[14] k. claffy, T. Monk, and D. McRobb. Internet Tomography. *Nature*, Jan 1999.

[15] I. Corporation. Intel DPDK - Data Plane Development Kit. `http://dpdk.org/`.

[16] D. M. Ritchie. A stream input-output system. *AT&T Bell Laboratories Technical Journal*, 63(8):1897–1910, Oct. 1984.

[17] L. Degioanni, F. Risso, and G. Varenni. PCAP Next Generation Dump File Format, Mar. 2004.

[18] J. Elischer and A. Cobbs. FreeBSD man pages - netgraph(4). `http://www.freebsd.org/cgi/man.cgi?query=netgraph&sektion=4`.

[19] L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9(6):733–745, Dec. 2001.

[20] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of internet hosts. *IEEE/ACM Trans. Netw.*, 14(6):1219–1232, Dec. 2006.

[21] D. Hartmeier. Design and performance of the openbsd stateful packet filter (pf). In *Proceedings of USENIX ATC 2002*, pages 171–180, Jun. 2002.

[22] B. Huffaker, D. Plummer, D. Moore, and k. claffy. Topology discovery by active probing. In *Symposium on Applications and the Internet (SAINT)*, pages 90–96, Nara, Japan, Jan 2002. SAINT.

[23] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, Jan. 1991.

[24] C. Inc. Cisco OnePK - Cisco One Platform Kit. `http://developer.cisco.com/web/onepk`.

[25] Internet Systems Consortium. Internet Systems Consortium: The ISC Domain Survey. `http://www.isc.org/solutions/survey`.

[26] M. Krasnyansky. Universal TUN/TAP device driver. `https://www.kernel.org/doc/Documentation/networking/tuntap.txt`.

[27] Y. Kuga, K. Cho, and O. Nakamura. On inferring regional as topologies. In *Proceedings of the 4th Asian Conference on Internet Engineering*, AINTEC '08, pages 9–16, New York, NY, USA, 2008. ACM.

[28] Y. Kuga, T. Matsuya, H. Hazeyama, K. Cho, and O. Nakamura. Etherpipe: An ethernet character device for network scripting. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, 2013.

[29] Y. Kuga, T. Matsuya, H. Hazeyama, K. Cho, R. Van Meter, and O. Nakamura. A packet i/o architecture for shell script-based packet processing. *Communications, China*, 11(2):1–11, Feb 2014.

[30] Lattice Semiconductor Corporation. LatticeECP3 Versa Development Kit, 2013.

[31] K. J. Lidl, D. G. Lidl, and P. R. Borman. Flexible packet filtering: providing a rich toolbox. In *Proceedings of BSDC'02*, Feb. 2002.

[32] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: An information plane for distributed services. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 367–380, Berkeley, CA, USA, 2006. USENIX Association.

[33] P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, k c claffy, and A. V. dat. The Internet AS-level topology: three data sources and one definitive metric. *SIGCOMM Comput. Commun. Rev.*, 36(1):17–26, 2006.

[34] T. Matsuya. TAP device driver for EtherPIPE. https://github.com/sora/ethpipe/tree/master/software/tappipe.

[35] S. McCanne and V. Jacobson. The bsd packet filter: a new architecture for user-level packet capture. In *Proceedings of USENIX'93 Winter Conference*, Jan. 1993.

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

[37] OpenFlowHub.org. Floodlight. http://floodlight.openflowhub.org/.

[38] Oracle Corporation. man pages section 7: Device and Network Interfaces dpli(7P). http://docs.oracle.com/cd/E19253-01/816-5177/dlpi-7p/index.html.

[39] D. Presotto and P. Winterbottom. The organization of networks in plan 9. Winter 1993 USENIX Conference Proceedings, 1993.

[40] R. Olsson. pktgen the linux packet generator. In *Proceedings of the Linux Symposium, Ottawa*, volume 2, pages 11–24, Jul. 2005.

[41] RIPE/NCC. RIPE Network Coordination Centre: RIPE Network Coordination Centre. http://ripe.net.

[42] L. Rizzo and M. Landi. netmap: Memory Mapped Access to Network Devices. *SIGCOMM CCR*, 41(4):422–423, Aug. 2011.

[43] A. Software. Vuze wiki - Auto Speed. https://wiki.vuze.com/w/Auto_Speed.

[44] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public internet measurement facility. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 17–17, Berkeley, CA, USA, 2003. USENIX Association.

[45] I. Starndard. IEEE Standard for Ethernet, Dec. 2012. IEEE 802.3(TM)-2012.

[46] tcpdump.org. TCPDUMP and LIBPCAP. http://www.tcpdump.org/.

[47] The DIMES project. The DIMES project: The DIMES project. http://www.netdimes.org/new/.

[48] The Open Networking Foundation. OpenFlow Switch Specification. Technical Report Version 1.3.1 (Wire Protocol 0x04), Sep. 2012.

[49] The Wireshark Foundation. Wireshark. http://www.wireshark.org/.

[50] Trema developers. Trema - Full-Stack OpenFlow Framework in Ruby and C. http://trema.github.com/trema/.

[51] University of Oregon. RouteViews Project. http://www.routeviews.org/.

[52] University of Washington: Computer Science and Engineering. iPlane: data. http://iplane.cs.washington.edu/data/data.html.

[53] University of Washington: Computer Science and Engineering. Scriptroute: undns. http://www.scriptroute.org/source/.

[54] O. N. Yohei Kuga, Kenjiro Cho and J. Murai. Visualization of asian as topology evolution. *JSSST*, 30(2):147–158, 2013.

Appendix A

Research History

Publications

Book

1. Akimichi, **Yohei Kuga**. Shape of the Internet. *Ohmsha*, ISBN:978-4-274-06824-9, 2011 (in Japanese).

Journal

1. **Yohei Kuga**, Kenjiro Cho, Osamu Nakamura, Jun Murai. Visualization of Asian AS topology evolution. *JSSST*, No.2, Volume 30, 2013 (in Japanese).

2. Takeshi Matsuya, **Yohei Kuga**, Rodney Van Meter, Shigeya Suzuki, Hideaki Yoshifuji, Jun Murai. Full pipelining for low-latency IP forwarding. *IEICE Trans. on Communications*, Vol. J96-B, No. 10, pp. 1095-1103, Oct. 2013 (in Japanese).

3. **Yohei Kuga**, Takeshi Matsuya, Hiroaki Hazeyama, Kenjiro Cho, Rodney Van Meter, Osamu Nakamura. A Packet I/O Architecture for Shell Script-based Packet Processing. *IEEE China Communications*, Vol. 11, No. 2, pp. 1–11, Feb. 2014.

4. Takeshi Matsuya, **Yohei Kuga**. PC Acceleration using FPGAs. *KEIO SFC journal*, Vol. 13, No. 2, 2014 (Invited paper, in Japanese).

Refereed International Conference

1. **Yohei Kuga**, Kenjiro Cho, Osamu Nakamura. On Inferring Regional AS Topologies. *ACM AINTEC: Proceedings of the 4th Asian Conference on Internet Engineering*, pages 37–38, New York, NY, USA, 2008. ACM.

2. **Yohei Kuga**, Kenjiro Cho, Osamu Nakamura. Comparison of Regional AS Topology Structures. *JSSST WIT workshop*, JP, 2012 (in Japanese).

3. Yuta Tokusashi, Takeshi Matsuya, **Yohei Kuga**, Jun Murai. Improving the Naturalness of Internet Video Conversation using a Low-Latency Pipeline. *DICOMO: Multimedia, Distributed, Cooperative, and Mobile Symposium.*, JP, 2013 (in Japanese).

4. **Yohei Kuga**, Takeshi Matsuya, Hiroaki Hazeyama, Kenjiro Cho, Osamu Nakamura. EtherPIPE: An Ethernet Character Device for Network Scripting. *ACM HotSDN: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 61–66, New York, NY, USA, 2013, ACM.

5. **Yohei Kuga**, Takeshi Matsuya, Hiroaki Hazeyama, Kenjiro Cho, Rodney Van Meter, Osamu Nakamura. Toward a Precision Network Scripting with a User-Programmable Dataplane. *USENIX NSDI Poster and Demo: 11th USENIX Symposium on Networked Systems Design and Implementation*, 2014, USENIX.

6. Takeshi Matsuya, **Yohei Kuga**, Hideaki Yoshifuji, Rodney Van Meter, Jun Murai. IP-NUMA for low-latency communication. *ACM CFI'14 Proceedings of The Ninth International Conference on Future Internet Technologies*, New York, NY, USA, 2014, ACM.

Miscellaneous Publication

1. Katsuhiro Horiba, **Yohei Kuga**, Hiroaki Hazeyama, Akira Kato. ACTIVE AND PASSIVE MONITORING AND ANALYSIS OF IP OPTION HEADER TRANS-

PARENCY FROM COVERT CHANNEL POINT OF VIEW. *APAN: PROCEEDINGS OF THE ASIA-PACIFIC ADVANCED NETWORK*, 2012.

2. **Yohei Kuga**, Takeshi Matsuya, Hiroaki Hazeyama, Osamu Nakamura. Toward flexible network tester with FPGA. *IEICE Technical committee on Internet Architecture 2012*, JP, 2012 (in Japanese).

Miscellaneous Activity

- Internship at LIP6 Networks and Performance Analysis group, UPMC Sorbonne UniversitÃl's (France, 2008, 3 month)
- Visiting the CAIDA research group based at the University of California's San Diego Supercomputer Center (USA, 2013, 2 week)