Future Generation Computer Systems 93 (2019) 687–701

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Implementation of a real-time network traffic monitoring service with network functions virtualization

Chao-Tung Yang [a], Shuo-Tsung Chen [b,c], Jung-Chun Liu [a], Yao-Yu Yang [a], Karan Mitra [d], Rajiv Ranjan [e,f,*]

[a] Department of Computer Science, Tunghai University, Taichung City, 40704, Taiwan, ROC
[b] Intelligence Recognition Industry Service Research Center (AIR-IS Research Center), National Yunlin University of Science and Technology, Yunlin 64002, Taiwan, ROC
[c] College of Future, Bachelor Program in Interdisciplinary Studies, National Yunlin University of Science and Technology, 64002, Taiwan, ROC
[d] Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Sweden
[e] School of Computer, China University of Geosciences, China
[f] School of Computing Science, Newcastle University, United Kingdom

HIGHLIGHTS

- This work customizes network service Neutron on OpenStack.
- The traditional managed switch is replaced by using Open vSwitch and a real-time traffic monitoring function is implemented with an IP-table filter.
- Since a NetFlow collector is built, there is no need to use hardware port mirrors to collect NetFlow data.
- This service integrates with OpenFlow to manage networking.
- The proposed network monitoring service in this work can be implemented in any kind of networking environments.

ARTICLE INFO

Article history:
Received 19 February 2018
Received in revised form 11 July 2018
Accepted 29 August 2018
Available online 24 September 2018

Keywords:
Software-defined networking
Network functions virtualization
OpenFlow
Virtualized switch
Network traffic monitoring

ABSTRACT

The Network Functions Virtualization (NFV) extends the functionality provided by Software-Defined Networking (SDN). It is a virtualization technology that aims to replace the functionality provided by traditional networking hardware using software solutions. Thereby, enabling cheaper and efficient network deployment and management. The use of NFV and SDN is anticipated to enhance the performance of Infrastructure-as-a-Service (IaaS) clouds. However, due to the presence of a large number of network devices in IaaS clouds offering a plethora of networked services, there is need to develop a traffic monitoring system for the efficient network. This paper proposes and validates an extensible SDN and NFV-enabled network traffic monitoring system. Using extensive experiments, we show that the proposed system can closely match the performance of traditional networks at cheaper costs and by adding more flexibility to network management tasks.

1. Introduction

The Network Functions Virtualization (NFV) extends the functionality provided by Software-Defined Networking [1] (SDN). It is a virtualization technology [2] that aims to replace the functionality provided by traditional networking hardware using software solutions [3–11]. Both NFV and SDN can be implemented using the OpenFlow technology [12,13]. On the one hand, SDN divides the traditional network hardware into two parts: the data plane and the control plane for efficient network configuration and management. On the other hand, NFV moves the services like firewall, WAN acceleration, load balancing and intrusion prevention system away from dedicated hardware into a virtualized environment. Thereby enabling, service providers to dynamically offer these services to their customers, with the ability to set up on-demand. Infrastructure-as-a-service (IaaS) is the most common service model in cloud computing, and network management is crucial in a large IaaS cloud computing environments. It is expected that the implementation of NFV can significantly enhance the performance of IaaS clouds [14]. By using virtualized network devices to replace traditional hardware, it becomes more convenient for network management in IaaS clouds. Besides the cloud environment, network virtualization can also be implemented in the traditional network environment. With the aim to enhance traditional network functionality, several studies on network virtualization [4–11] targeted the traditional networking hardware, such as firewalls, load balancers, routers, and managed switches.

* Corresponding author at: School of Computing Science, Newcastle University, United Kingdom.
E-mail addresses: ctyang@thu.edu.tw (C.-T. Yang), raj.ranjan@ncl.ac.uk (R. Ranjan).

https://doi.org/10.1016/j.future.2018.08.050
0167-739X/

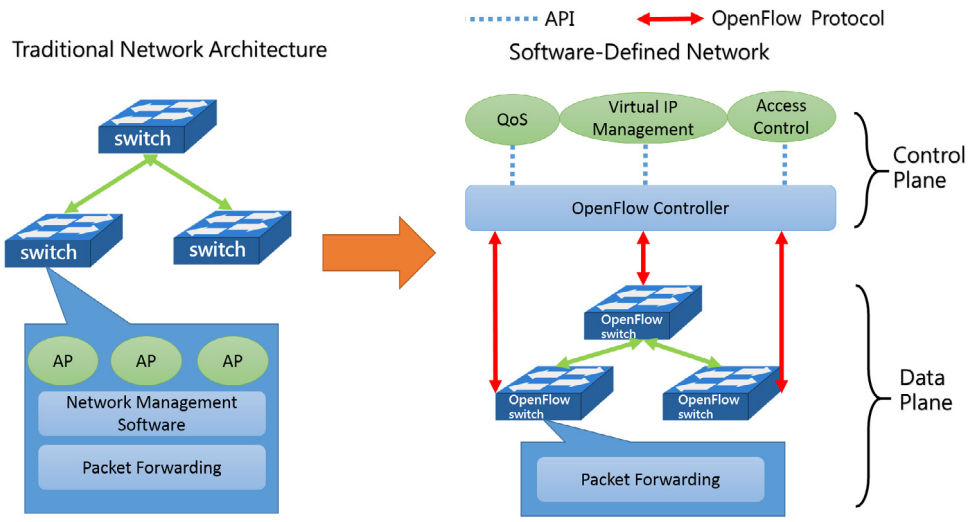

Fig. 1. Traditional network architecture mapping to a software defined network.

OpenStack, a well known open source cloud operating system that has become very popular in recent years [15–18]. OpenStack contains a core network service called Neutron that provides network services for the instances running on OpenStack. Neutron is based on Open vSwitch and supports several applications and plugins. In this paper, we raise the following question: "*if Neutron can be used to provide network services for virtual machines, why not to use it to provide same services for traditional physical machines?*" Therefore, in this paper we used Neutron to implement a real-time traffic monitoring system and compared its performance with a similar hardware system.

Our system aims to provide network administrators an easy-to-build network traffic monitoring system without additional hardware costs such as those related to routers and switches. This is achieved by providing routing functions to replace traditional layer 3 (L3) network hardware devices. In particular, in the Neutron architecture, we set an iptable filter in network namespace to implement a real-time network traffic calculator. Also, NetFlow configuration in Open vSwitch is set to send NetFlow data that is used for network analysis. Further, the OpenFlow controller is used to manage the entire service. We also implement a virtual switch and establish a traffic monitoring system with OpenFlow to manage networks. We used the SDN to build a system that has a traffic monitoring functionality as well as functions of a physical switch. In other words, we focus on virtualization of essential functions of a physical switch to implement a network traffic monitoring system. The proposed system is extensible to various other network environments that can assist the network administrators to monitor and manage networks. The proposed system can be easily combined with conventional computers, thereby eliminating the need to buy additional network hardware.

2. Background

2.1. NetFlow

The NetFlow [19] functionality incorporated in the Cisco router is used to analyze network traffic and generate data. The network administrators can use the data provided by NetFlow to determine network usage. The NetFlow version 5 data consists of network packets that define seven values: 1. Ingress interface, 2. Source IP address, 3. Destination IP address, 4. IP protocol, 5. Source port, 6. Destination port, 7. IP type. The classic NetFlow architecture incorporates three elements to monitor the network. These include Flow exporter, Flow collector, and Analysis application. The Flow exporter generates the flow data; this data is sent to the Flow collector for storage and analysis.

2.2. Open vSwitch

In this paper, we choose the Open vSwitch [20] which is a virtual switch based on open source technology [21,22]. Open vSwitch provides protocols to communicate with OpenFlow controller, so it is an essential element to implement the SDN. Running virtual switches can provide OSI Layer 2 network communication for the virtual machine in IaaS clouds as it can be treated as a software switch by network administrators.

2.3. OpenFlow

OpenFlow [23] is an open standard protocol to control the packet flow over the network. It divides traditional switch into two parts, i.e., data plane and control plane. In the traditional networking devices, the forwarding plane (data plane) and routing plane (control plane) are on the same device. The network administrators must learn to control different devices as these are sold by several vendors, each having their own rules and standards. Therefore, it is complicated and inconvenient to set up a configuration successfully. A network administrator will, therefore, need considerable knowledge of most of the present device types. In comparison to traditional network devices, OpenFlow gives a new way for network administrators to manage networks: using a controller to control routing rules, and hardware only does packet forwarding. OpenFlow Controller has many distribution versions. In this paper we consider the OpenDayLight [24] version, OpenDayLight provides several plugins that can cater to several requirements posed by the network administrator. For instance, in this paper, we incorporate OpenDayLight plugin with OpenStack Neutron.

2.4. OpenStack

OpenStack [25] is a popular cloud operating system that comprises three main components: computing, networking and storage. The compute component offers service like deploying virtual machines and managing images. Networking component helps each component and virtual machine to communicate. Storage component offers users or virtual machines different type of storage space to manage. These components in conjunction are used to build an IaaS cloud. OpenStack provides an easy-to-use dashboard and APIs to manage the IaaS components.

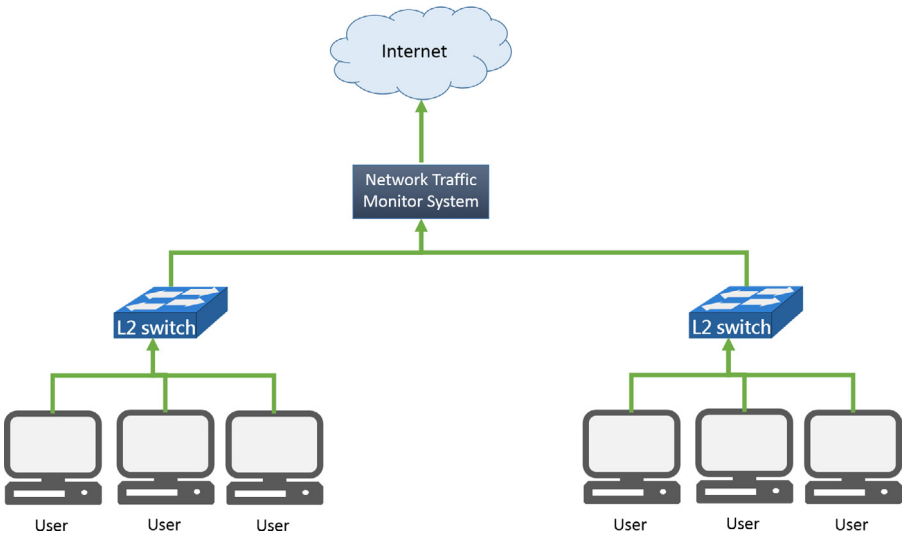

Fig. 2. New network environment.

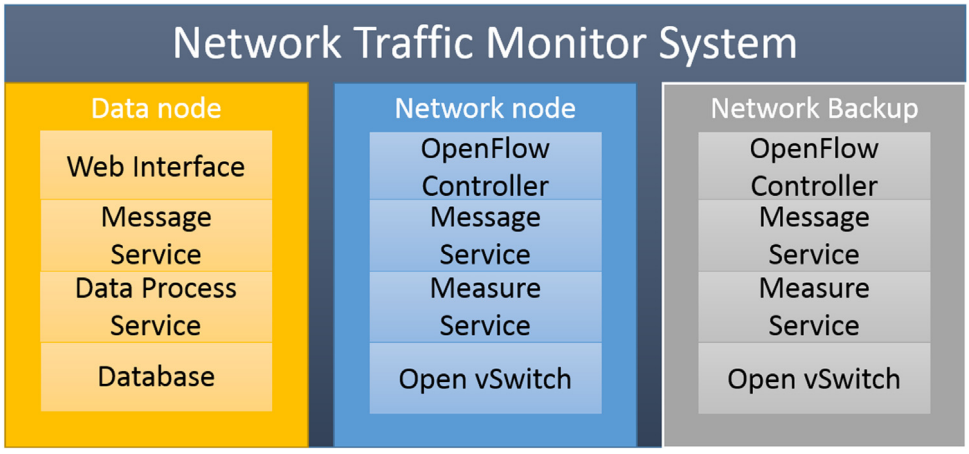

Fig. 3. System architecture.

2.4.1. Neutron

Neutron [26] is a network service that provides OpenStack instances network capabilities. It is based on Open vSwitch and is combined with three components: ML2 plugin, L3 agent, and the DHCP agent. The ML2 plugin provides network VLAN and network tunnel for compute nodes. L3 agent builds a virtual router in network service and DHCP agent acts as a virtual DHCP server.

3. System design and implementation

In this section, our system design and architecture are first presented. Next, the implementation of the proposed system is introduced.

3.1. System architecture

In the traditional network environment, the traffic monitoring system usually uses the physical switch as shown in Fig. 1. In this environment, to retrieve NetFlow data for analysis, the physical switch should be able to support the NetFlow protocol. Our goal is to re-model this system architecture by using the virtualization technology, since this approach can greatly improve flexibility for network management. In this article, we describe the design of a network traffic monitoring system that incorporates a virtual switch, an OpenFlow controller, and a traffic measuring service. The system consists of three nodes: one data node, one network node and one network node backup to enable high availability (HA) of the system. Network traffic is entered using the virtual switch. The measuring service calculates the traffic, and then the collected data are sent by a message service to the data node. A data input service sends the data to database. When the data input service detects problems such as malware activity, policy violation, protocol anomaly, virus detection, bandwidth anomaly, etc. in data, it uses an application program interface (API) in the OpenFlow controller to control the virtual switch. Fig. 3 illustrates the system architecture. The network node uses the message service to examine each other. If the master node is offline or fails, then a slave node replaces it. This ensures that the Internet usage through the network node will not be interrupted. The network administrator can use a web interface to assess the state of the network (see Fig. 2).

3.2. Design flow

3.2.1. Network node

To implement virtualization of the network function, we design a network node (Fig. 4) that is used to control the virtual switch and calculate network traffic. This system is based on the concept of SDN using OpenDayLight architecture. The network node is used to perform four services. When the network traffic enters the virtual

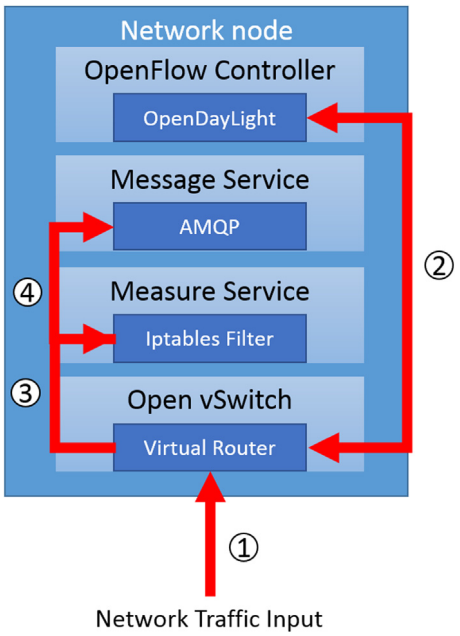

Fig. 4. Network node architecture.

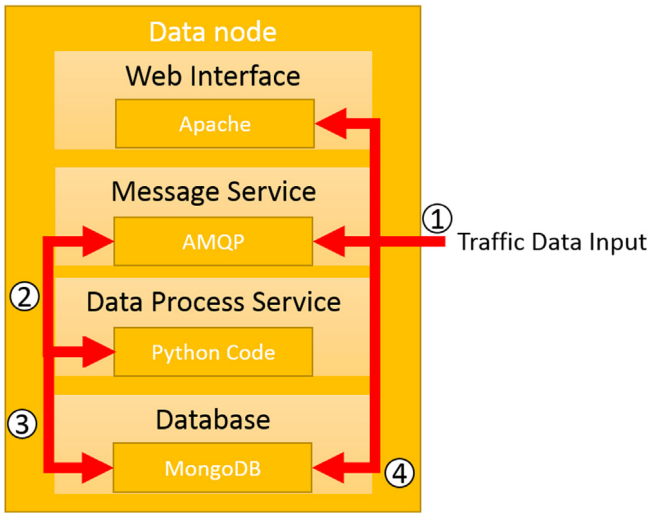

Fig. 5. Data node architecture.

Table 1
Data collections format.

Title	LastUpdate	CheckTime	Data1	Data2
Unit	timestamp	Seconds	Bytes	Packets

router, the first service initially manages it, i.e., Open vSwitch, and subsequently detected by the controller rules of OpenFlow (OpenDaylight), i.e., the second service. When no problem is detected, traffic is allowed to go through Open vSwitch and transferred to a gateway in a physical network environment. The third service involves calculating network traffic in the process by using defined rules. The filter table of iptables is used to calculate the number of network bytes and whether the packet is for input or output. All data are then sent to the fourth service, i.e., the message service. Each statistical data item is stored in a data node by the message service according to the Advanced Message Queuing Protocol (AMQP). To calculate traffic amount, the amount of data throughput in the filter table at a check time is assessed. The measuring service defines the check time. Different times may give rise to varied results, which we will discuss in the experimental section.

3.2.2. Data node

The data node is used to process data collected from the network nodes and to perform four services: the web service, data processing service, message service, and database service (Fig. 5). The data processing service is employed to format data. Data sent by the message service are obtained according to the AMQP. Table 1 shows the data collection format. The data processing service is also used to detect problems in data. When detecting data with problems, the data processing service notifies the OpenFlow controller API, which manages Open vSwitch. The OpenFlow controller can then limit source data traffic to prevent malicious intrusions. Then the processed data are sent to database. These data are used to display or analyze in the web interface. The data processing service is coded in Python. In this process, data must be frequently written and read. Therefore, we use the NoSQL database architecture (MongoDB).

3.2.3. Network node

We use the OpenStack Neutron L3 solution to back up the network. The primary technology used is the Virtual Router Redundancy Protocol (VRRP). The system architecture is shown in Fig. 6. Two nodes using the message service maintain the connection, each ensuring that the other is alive. When the network traffic enters, it is first assessed by the master node. When the message service nodes cannot detect each other, the network traffic is transferred to the slave node. Using VRRP ensures that the service is not interrupted when a user uses the network service. Implementation of VRRP can be divided into two parts: creating a virtual router between two nodes and replacing a failed master node with a slave node.

The first part of the VRRP solution is to create a virtual router between two nodes, as shown in Fig. 7. The master node uses Open vSwitch to create the virtual router, high availability (HA) network, and HA port. The Network agent builds interface and HA configuration file, and then sends messages to the slave node. The slave node gets the message and is set with the same configuration as the master node and then keeps the connection with the master node.

The second part of the VRRP solution is to enable the slave node to replace the master node when it fails, as shown in Fig. 8. When the virtual router in the master node fails or is offline, the network agent will send the message to the slave node. The slave node starts up a virtual router according to configuration files. If the network agent in the master node fails or is offline, the slave node will automatically start up the virtual router when it cannot detect connection of the master node.

3.3. Monitoring mechanism

The traffic monitoring mechanism in the proposed system is divided into two parts: real-time monitoring function and statistical data analysis. The real-time monitoring function combines with the OpenFlow controller so that the system will able to use APIs to manage traffic entering into the system. Statistical analysis of traffic data is displayed by NetFlow. Network administrators can use these real-time monitoring function to detect network behavior anomaly and statistical data to analyze the network traffics.

3.3.1. Real-time traffic monitoring function

One way to implement a real-time traffic monitoring function is to use the network namespace as shown in Fig. 9. OpenFlow System uses ip netns to create a network namespace so that

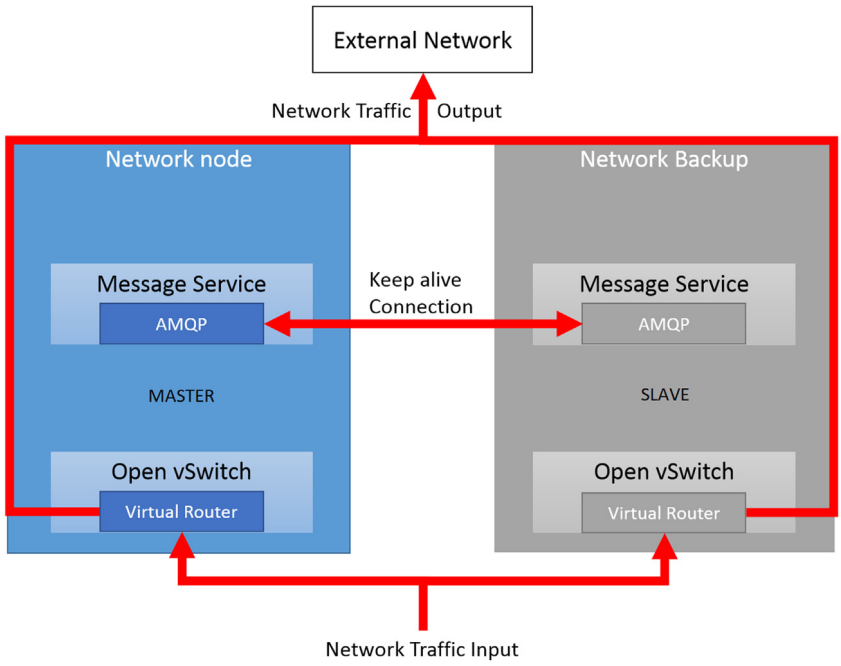

Fig. 6. Network backup architecture.

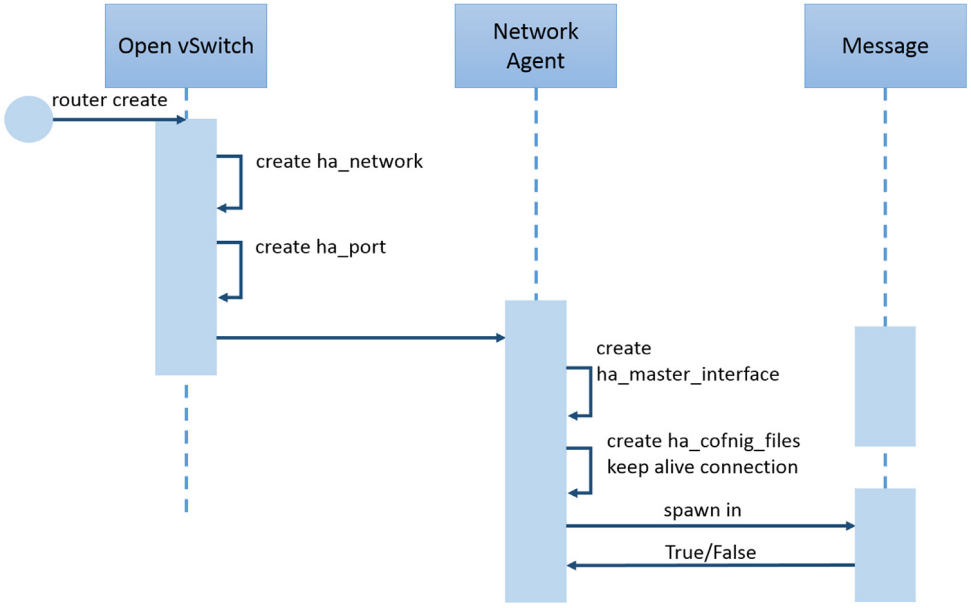

Fig. 7. Virtual router creation.

namespace inherits configuration from the host. The traffic will come into specific namespace when the namespace is used to create a virtual router.

Namespace creates rules in the iptable filter. If the source IP of coming traffic is matched with rules then this traffic data will be collected. Collected data will be periodically sent to the data node for administrators to analyze. The work flow of the filter table calculation mechanism is shown in Fig. 10

3.3.2. Statistical data analysis

The statistical data analysis is implemented in the way similar to that of implementing real-time traffic monitoring function. The system uses the ip netns command to create the network namespace and virtual router. Open vSwitch supports the NetFlow protocol to collect all packet information through virtual routers. This packet information will be translated as NetFlow data and sent to the data node that periodically collects data by means of flow-capture as shown in Fig. 11.

3.3.3. Calculate mechanism

In the proposed system, network monitoring and traffic measurement are the two most important functions. For network administrators to easily manage networks, a limit value of K is set for reference in real-time traffic monitoring. When real-time traffic is bigger than K, the OpenFlow controller will perform some actions on this traffic flow.

$$Limit = K \tag{1}$$

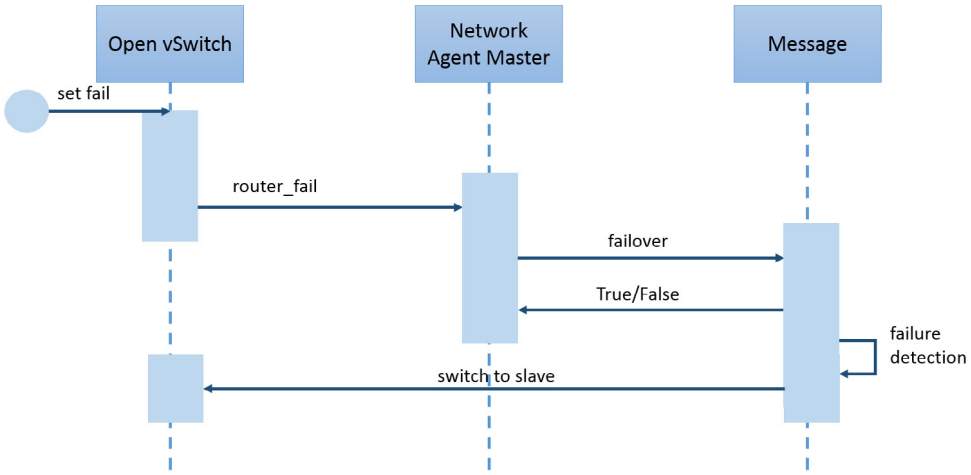

Fig. 8. Network node failure.

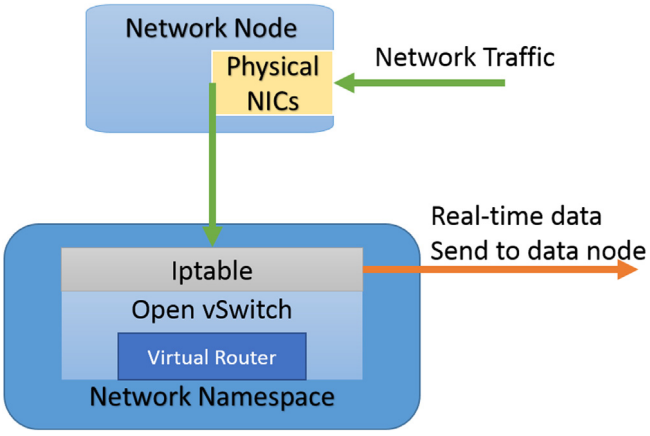

Fig. 9. Network namespace in network node.

The real-time traffic monitoring unit calculates the data throughput within a checking time interval which is defined based on measuring service. However, different checking time may obtain different results. Eq. (2) is used to calculate number of traffic bytes statistically (i.e., traffic flow) and Eq. (3) is used to calculate statistical packets. $Data_b$ is traffic flow data in bytes, T_b is packets bytes, T_p is packets time and T_c is checking time.

$$Data_b = \frac{T_b}{T_c} \tag{2}$$

Table 2
Data collections format.

Title	Src_ip	Dest_ip	Prot	Src_port	Dest_port	Octets	Packets
Unit	ip	ip	protocol	port_number	port_number	Bytes	Packet

$$Data_p = \frac{T_p}{T_c} \tag{3}$$

After receiving real-time data, the data node formats the data to help network administrators easily monitor the network environment. The used format is shown in Table 1. The table includes last-updating time, checking time, bytes, and packet information.

For statistical analysis, we use the NetFlow data format. The used version of NetFlow data is version 5, which has been widely adopted. As shown in Table 2, the format includes source IP, destination IP, protocol type, source port, destination port, bytes, and packets.

We can set K according to statistical data in real-time traffic service. For example, K can be set according to statistics of a single-day data. We can treat single-day statistical data as a normal distribution, and after getting rid of outliers, calculate its mean to set K. In the real-time traffic monitoring unit, two values are collected: bytes and packets. When network traffic has abnormal events, it may cause exception values in bytes or packets. So K can be set according to statistics of either one of them.

We considered the "68–95–99.7" rule where 68.27%, 95.45% and 99.73% of the values lie within one, two and three standard deviations of the mean, respectively. Assuming that the data in our

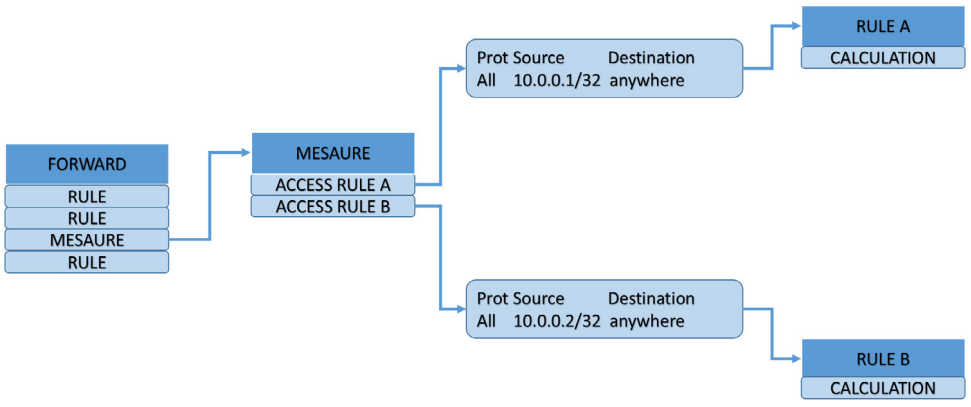

Fig. 10. Filter table calculation mechanism.

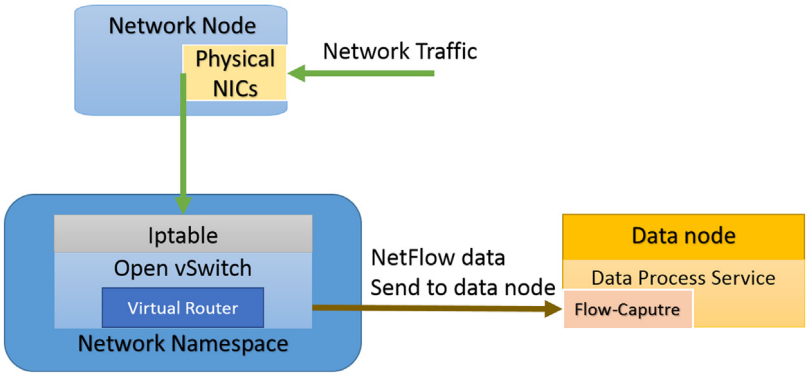

Fig. 11. NetFlow data collect mechanism.

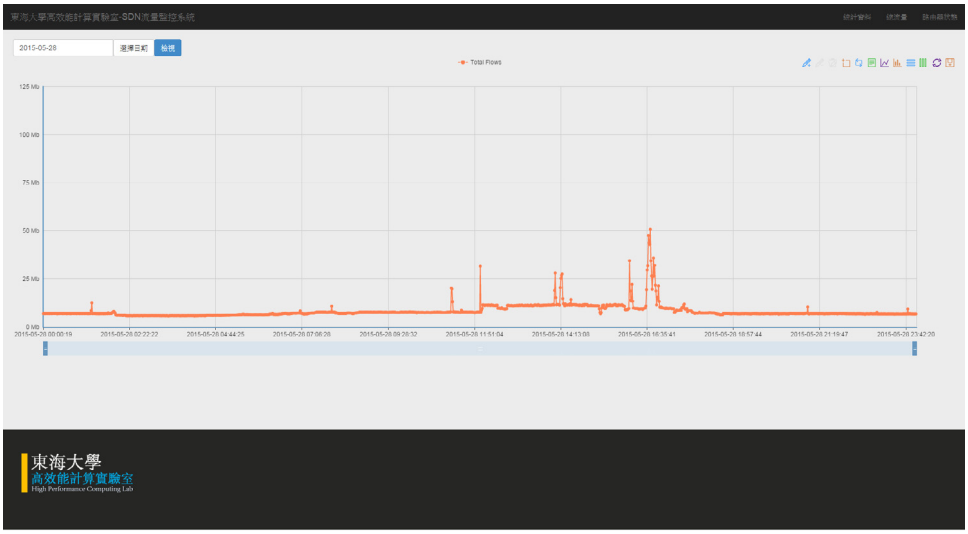

Fig. 12. Web management—flow state.

data set are normally distributed, 99.7% of the data will be within +/3 standard deviations from the mean for outlier value [27].

$$\mu_b = E(X_b) = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{4}$$

$$\mu_p = E(X_p) = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{5}$$

First, we suppose set X_b and set X_p represent statistics of bytes and packets, respectively. Based on these two sets, we obtain two means: μ_b and μ_p.

$$Var(X_b) = E[(x-\mu_b)^2] = \frac{1}{N}\sum_{n=1}^{N}(x_n-\mu_b)^2 \tag{6}$$

$$\sigma = \sqrt{Var(X_b)} \tag{7}$$

$$outlier = O_{nb} = \frac{x_n-\mu_b}{\sigma} \tag{8}$$

$$Var(X_p) = E[(x-\mu_p)^2] = \frac{1}{N}\sum_{n=1}^{N}(x_n-\mu_p)^2 \tag{9}$$

$$\sigma = \sqrt{Var(X_p)} \tag{10}$$

$$outlier = O_{np} = \frac{x_n-\mu_p}{\sigma} \tag{11}$$

We further obtain values of the variance, standard deviation and outlier. The purpose is to get an accurate mean, which can be used to set K for the real-time traffic monitoring system.

Algorithm 3.1: ESTABLISH NEW STATISTICAL DATA SET(BYTES)(X_b, O)

comment: Remove outliers value from Statistical Data Array $X_b[i]$

comment: $O_b[i]$ is a outliers array

for $n \leftarrow 1$ **to** *length of* X_b

do { **if** $-3 \le O_b[i] \le 3$ **then** $\{X_b[i] \leftarrow X_b[i].val.del$

Algorithm 3.2: ESTABLISH NEW STATISTICAL DATA SET(PACKETS)(X_p, O)

comment: Remove outliers value from Statistical Data Array $X_p[i]$

comment: $O_p[i]$ is a outliers array

for $n \leftarrow 1$ **to** *length of* X_p

do { **if** $-3 \le O_p[i] \le 3$ **then** $\{X_p[i] \leftarrow X_p[i].val.del$

We drop two outlier values O_{nb} and O_{np} and recalculate means for the two data sets X_b and X_p. Now we obtain two means that can

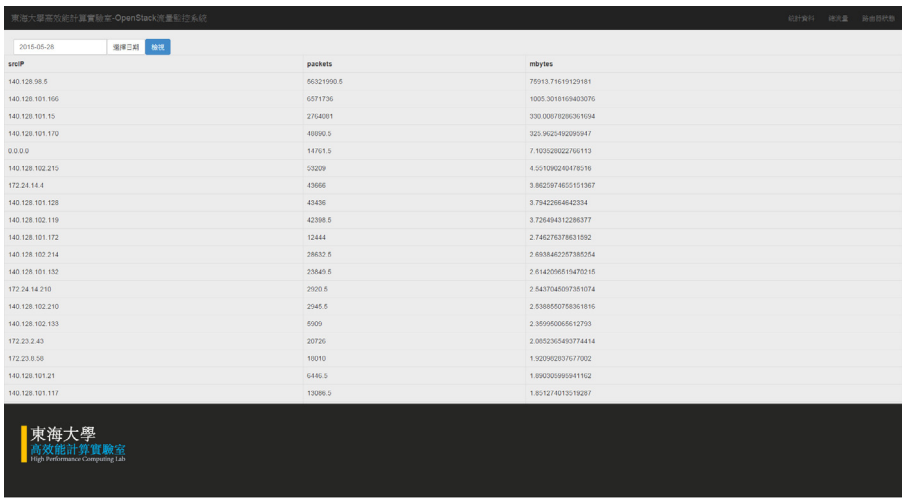

Fig. 13. Web management—flow data.

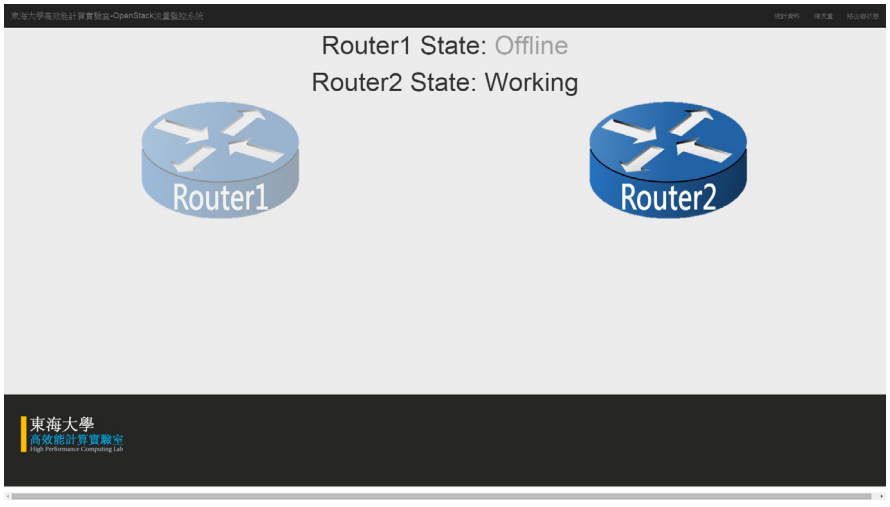

Fig. 14. Web management—router state.

be used to set K, i.e., two values of K are decided, one according to bytes data Eq. (12), the other according to packets data Eq. (13).

$$\mu_b = E(X_b) = \frac{1}{N}\sum_{n=1}^{N} x_n = K_b \tag{12}$$

$$\mu_p = E(X_p) = \frac{1}{N}\sum_{n=1}^{N} x_n = K_p \tag{13}$$

3.4. Web management

To help the network administrator efficiently manage the network, we design a web management page. This web page consists of three functions: the real-time flow state monitoring, statistical data query, and virtual router status. The web management page in default shows real-time traffic data on the index page, which will automatically load real-time traffic data collected on the current day. According to configuration settings, data will be written into the database every minute. The network administrator can obtain those data from this page. The network administrator can also select the date by using the calendar function. The observing time, shown in the *x*-axis, can be zoomed in or out as required.

Table 3
Hardware specification.

Host name	CPU	Memory	NIC
Data node	Intel(R) Xeon(R) E5645	8 GB	1 Gb
Network node	Intel(R) Xeon(R) E5645	8 GB	1 Gb
Network backup	Intel(R) Xeon(R) E5645	8 GB	1 Gb

Table 4
Software specification.

Software	OpenDayLight	Open vSwitch	OpenStack	Python
Versopm	Helium	2.0.2	Juno	2.7.6

As shown in Fig. 12, the statistics data query page offers information collected from NetFlow capture over the day. Every day the captured data are sent to database at 12.00 am. And those data can be queried by using this query page, which automatically displays top 20 sources IPs on this page after sorting all data. The network administrator can reference this information to adaptively set the value of K for the monitoring system.

The router status page displays states of routers: indicating which router providing network services, and showing the other router being at work, standby, or off-line as shown in Fig. 14.

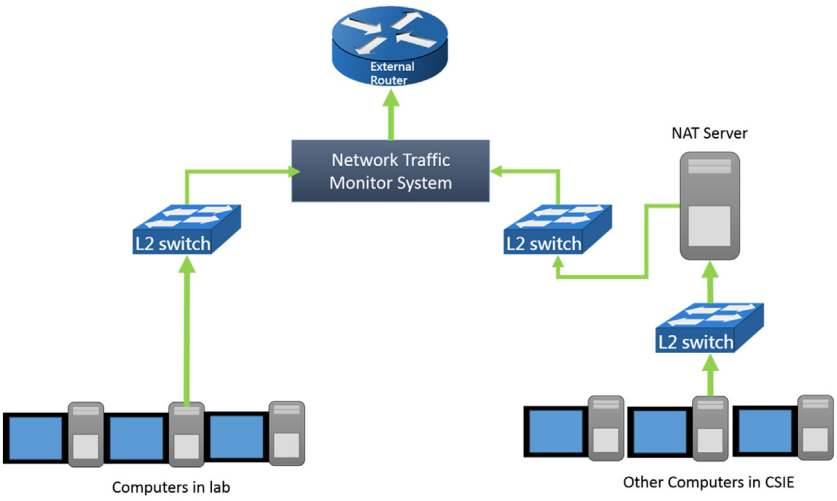

Fig. 15. Experimental architecture.

The network administrator can use this page to quickly tell the operation status of the whole network.

4. Experimental environment and results analysis

This section first presents the experimental environment for the proposed system design. Then, we implement the proposed system and obtain several useful results based on this experimental environment.

4.1. Experimental environment

4.1.1. Hardware

The experimental environment consists of three computers with same hardware specifications: 12-core CPU, 8 GB memory, and 1 Gb Network interface cards, as listed in Table 3 .

4.1.2. Software

Table 4 list the software used in our experiments. We used Helium version as the OpenFlow controller. We used Open vSwitch ver. 2.0.1, and used OpenStack with ver. Juno. Lastly, we used Python ver. 2.7.6.

We have built the proposed system in a real network environment. This system captures traffic of computers connecting to the Internet and manages IPs of computers. The experimental architecture is shown in Fig. 15.

4.2. Experimental results

We designed and conducted several experiments to test the performance of our proposed system. First, we tested the virtual router loading capacity as shown in Fig. 13. Then the performance comparison between the virtual and physical routers was conducted. Various TCP/UDP packet sizes were used as input to test its performance. The network response times were measured. Later we tested the accuracy of the monitoring system, consisting of real-time measurement and statistical data analysis. We used different database architectures to store data and compared their data processing speeds. Finally, we tested the performance of the high availability (HA) mechanism in this system.

4.2.1. Virtual router experiments

We used network testing software iperf for performance measurement of the virtual routers. Our experimental testbed consists of the client and server. A large number of packets were sent by the client to perform the stress test of the network. We used two protocols, i.e., TCP and UDP and considered six packet sizes of: 64 bytes, 128 bytes, 256 bytes, 512 bytes, 1024 bytes, and 2048 bytes as shown in Figs. 17 and 18, to compare performance of the virtual router and physical router as shown in Fig. 14. Fig. 16 shows results of the TCP experiment. From the bar chart, we observe that when the packet size increases, the network speed increases. The maximum speed appears with the packet size of 2048 bytes.

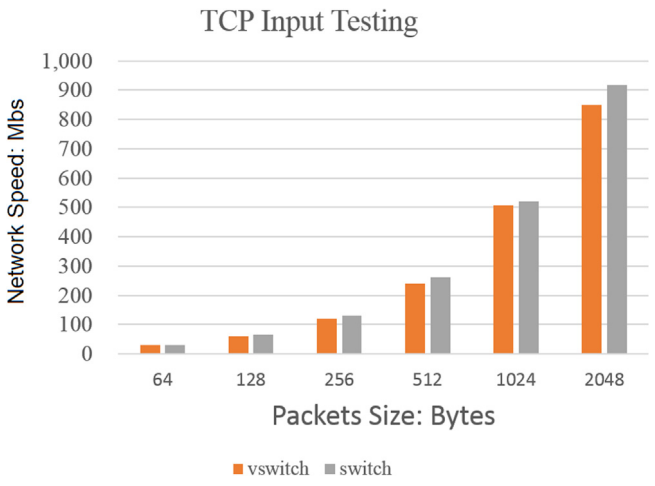

Fig. 16. TCP input testing.

As mentioned previously, the hardware specification of the network interface is 1 Gb per second. From the line chart, we can find the maximum speed of the physical router is nearly 90% of 1 Gb, and that of the virtual router speed is nearly 85% of 1 Gb. We conclude that performance of hardware router is better than that of the virtual one. However, since performance of virtual router is nearly 85% of the network interface, we consider it sufficient if the network usage is not significant.

Fig. 18 shows results of the UDP experiment. From the bar chart, we observe that the maximum speed appears in packets with the size of 1024 bytes. When the packet size increases to 2048 bytes, speeds of the physical and virtual router are slowed down. It may be caused due to the maximum transfer unit (MTU). UDP is simpler protocol than TCP, and it we cannot record lots of packet information.

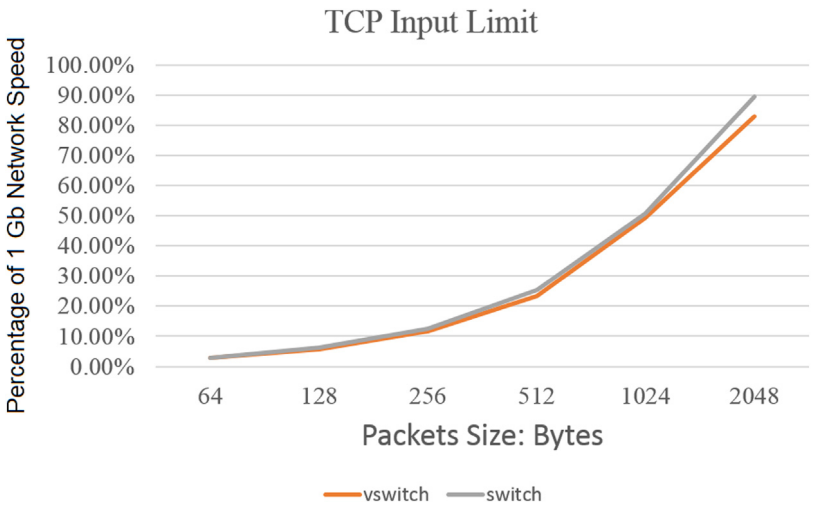

Fig. 17. TCP input limit.

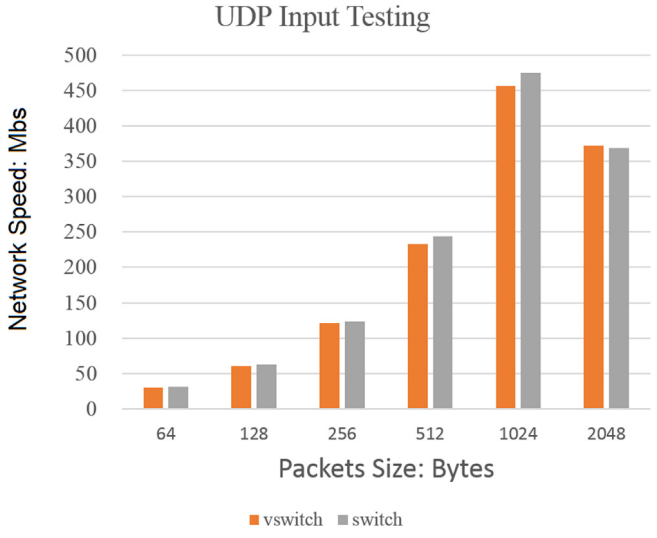

Fig. 18. UDP input testing.

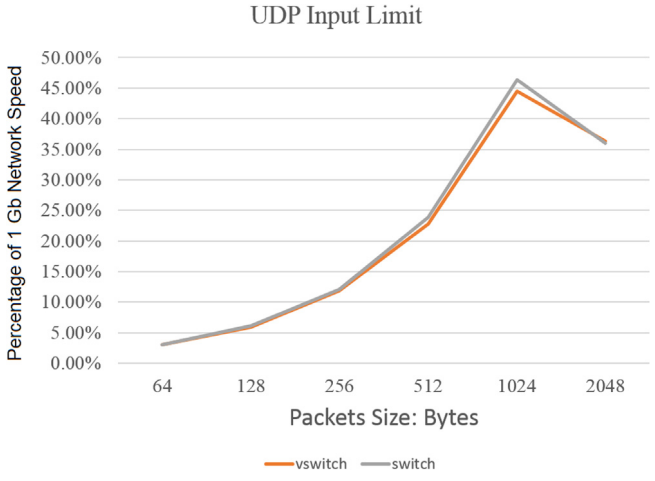

Fig. 19. UDP Input Limit.

From the line chart, we observe that in case of UDP, the speeds of both virtual and physical routers have little difference, approximately 5%. This result means that when UDP is used to transfer packets, the user cannot tell the difference between the virtual and physical routers (see Fig. 19).

Besides testing transmission speed of TCP and UDP, we also tested latency of the network, as shown in Fig. 20. This experiment lasted 5 min. We pinged to test network latency time per second; the target was the Chunghwa Telecom Domain name server. We observe that the trend of the latency of the virtual or physical network was similar, and long latency times have been spotted in both of them.

4.2.2. Real-Time measuring experiments

To test the accuracy of the real-time measurement system, we considered three time periods: 10 s, 30 s, and 60 s to write the same file to the database. The results are shown in Fig. 21. From our results, we found that when the data write frequency is higher, the accuracy is higher as well. However, writing database with high frequency may cause high traffic loading. From the line chart in Fig. 22, we observe the accuracy is almost 99.64% when measured once per minute; and 99.82% measured once per 10 s. We decide to use the minute period in our measurement system configuration.

4.2.3. Database experiments

Because lots of data are written into the database in our system, the performance evaluation of database system is critical. We compared two databases of different architecture: relational database and NoSQL. To test their performance, we used raw data of large amount: 240,000 bytes, 1,620,000 bytes, 6,370,000 bytes, 26,100,000 bytes, and 38,650,000 bytes. Each raw data set was written into both databases and its processing time was observed. The experimental results are plotted in Fig. 23. We observe that the processing time of NoSQL is longer than that of the relational database when the data size is more than 1,620,000 bytes. From the line chart, where y-axis represents amount of processed raw data per second, we observe that when the amount of data is 240,000 bytes, the processing speed of NoSQL is faster than that of relational database; but when the size of raw data increases to 1,620,000, the processing speed of relational database is faster than that of NoSQL. Moreover, as the size of data increases, the growth rate of processing speed of relational database slows down, and processing speed NoSQL almost remains constant. We chose NoSQL to implement the database in our system since, in our network environment, the amount of data in a day is close to 240,000 bytes (see Fig. 24).

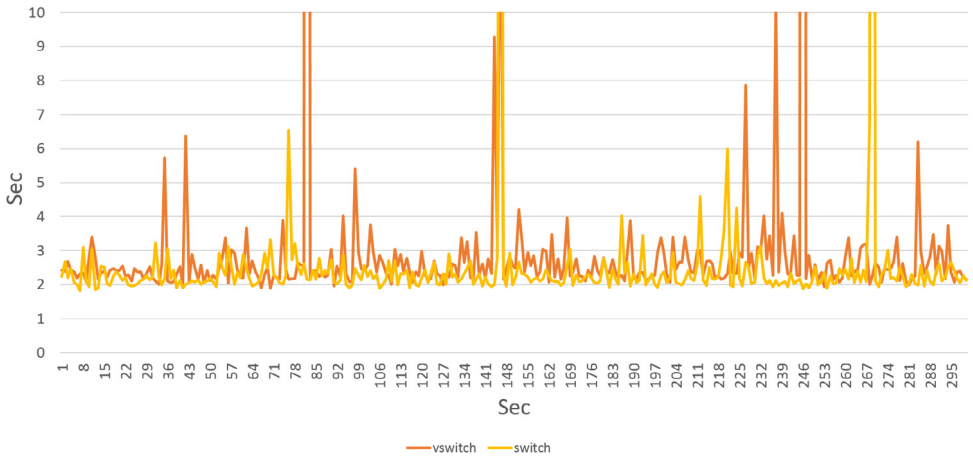

Fig. 20. Response time testing.

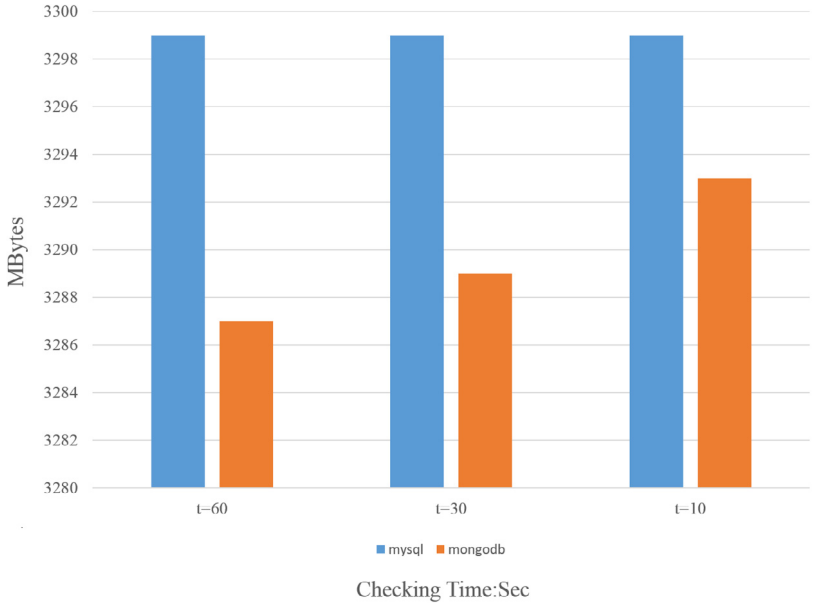

Fig. 21. Real-time measuring testing.

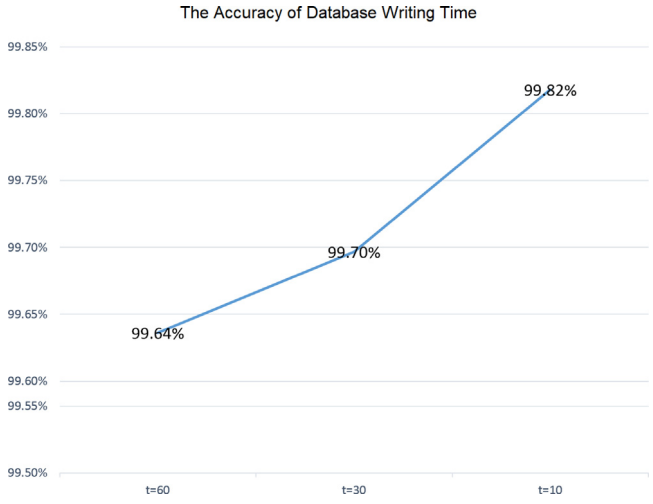

Fig. 22. Real-time measurement testing.

4.2.4. NetFlow data experiments

We conducted experiments to verify the accuracy of the NetFlow data collected from the virtual router. We compared these data with the data collected from the Cisco router. Fig. 25 shows the results. We observe that data collected from the Cisco router are similar to raw data, whereas data collected from the virtual router show difference. As shown in Fig. 26, the relative error in the Cisco router is less than −1%, but that in the virtual router is almost 12%.

The data collected by the NetFlow can be used to set the limit value K. Fig. 27 shows 7 days statistics data from the virtual router for our department. The unit of the *y*-axis is amount, and the *x*-axis is Mbps. We observe that most flow is between 4 and 7 Mbps. We may set limit K as 15 Mbps for this statistics data. When an IP's flow is greater than this value, there is some abnormal activity in it. Then network administrator can treat this IP as an observation target.

4.2.5. Network high availability experiments

We performed experiments to test the availability and response time of the high availability (HA) function of the network system. We used iperf to test the network. We stopped the master network

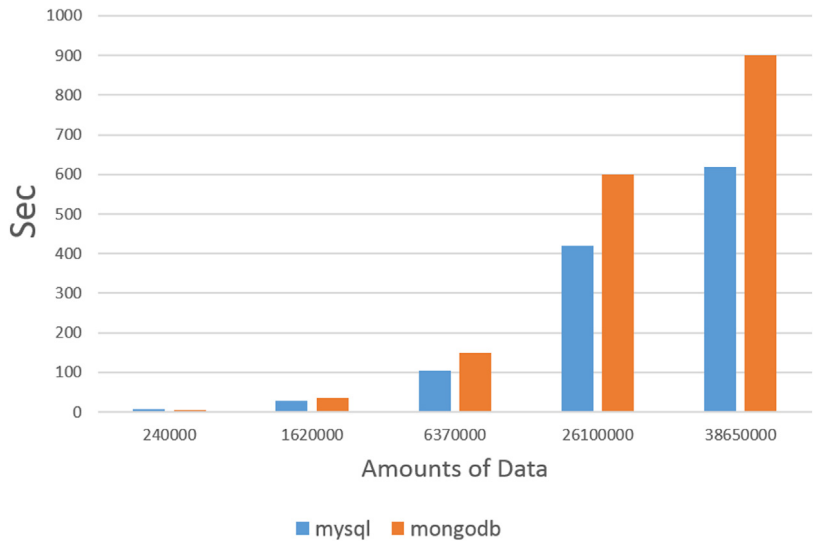

Fig. 23. Database testing.

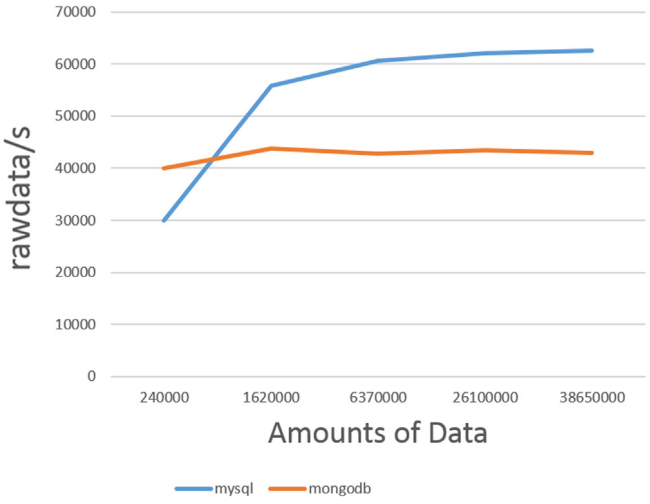

Fig. 24. Database input testing.

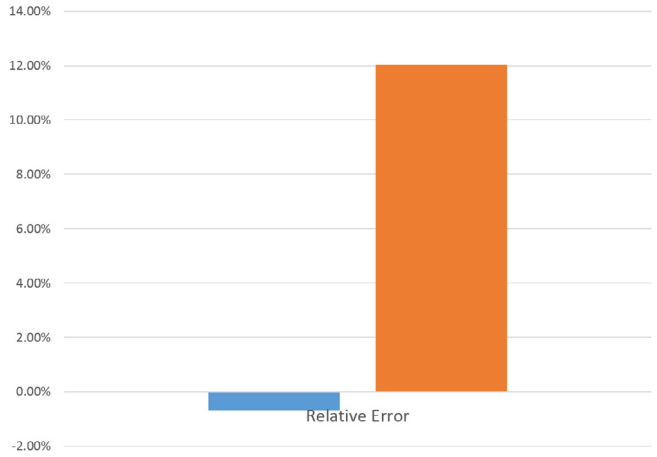

Fig. 26. NetFlow data measurement testing.

node and observed the reaction of the network flow. Figs. 28 and 29 shows the experimental results. We repeated this experiment several times, for which we observed the best response time of 3 s and the worst response time of 13 s. From the figure, we observe that the network flow reaches 0 at time 5 s of time of the experiment because the master network node was shut-down at that time. In the best case scenario, the network flow came up to the normal value at time 8 s, which means network HA was working successfully. In the worst case scenario, the master network node was stopped at time 4 s. Later, the network flow comes up to the normal value at time 17 s, thus, the response time of HA was 13 s.

4.2.6. OpenFlow controller experiments

We performed experiments to verify the effectiveness of the network management mechanism. Fig. 30 shows the experimental

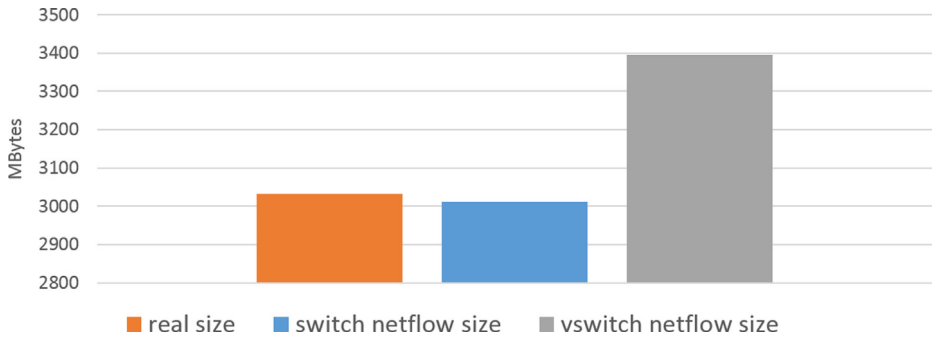

Fig. 25. NetFlow data measuring testing.

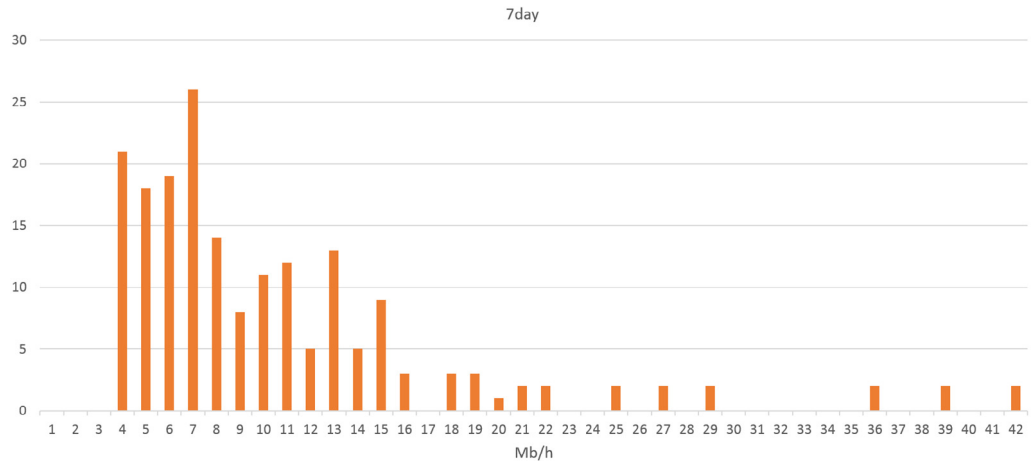

Fig. 27. NetFlow statistics data.

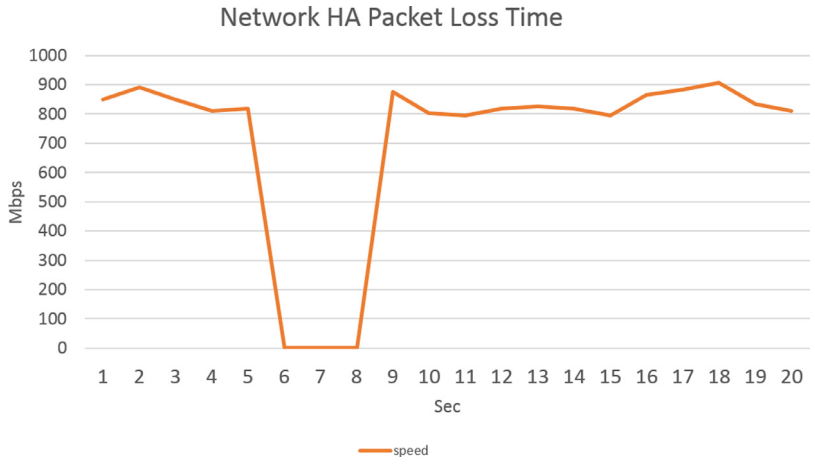

Fig. 28. Network high availability.

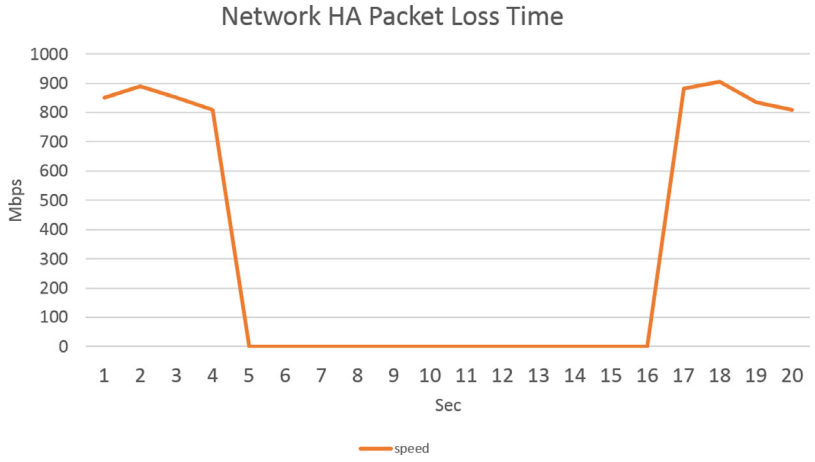

Fig. 29. Network high availability.

results. A personal computer was used to download a file from Web via the virtual router; when download speed is greater than the limit value K, the real-time monitoring system called the OpenFlow controller to deny IP of this PC. From the figure, we observe that the peak speed is about 50 Mbps. We set K as 50 Mbps at time 6 s. We observed that the flow decreases at time 7 s, and it comes down

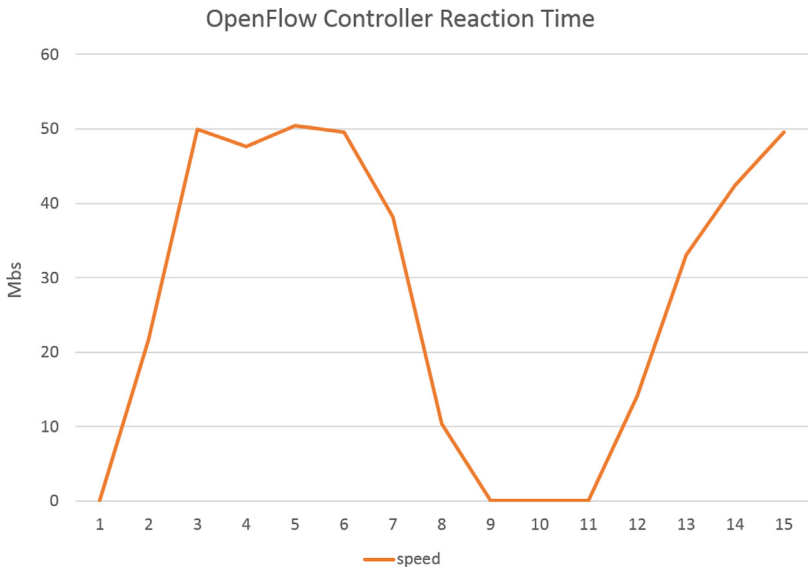

Fig. 30. OpenFlow controller reaction time.

to 0 at time 9 s. We canceled K and deleted the IP from the denial list at time 11 s, and observed that the flow comes up to 50 Mbps again. The experimental result shows that the OpenFlow controller is effective in managing the virtual router and its response time is very fast as shown in Fig. 30.

5. Conclusion and future work

This work uses software-defined networks and network functions virtualization to implement a network traffic monitoring system and compares its performance with that of the system with traditional hardware architecture. From the experimental results, we find that although the proposed system cannot out-perform the system with traditional hardware architecture, the proposed system can closely match the performance of traditional networks at cheaper costs and adds more flexibility to the network management tasks. The implemented system fulfills our initial goals: to use concepts of network functions virtualization and software-defined network to build a network traffic monitoring system and make it easy for network administrators to manage networks. As the future work, we aim to test the proposed system in the broader network environment. We also intend to test our system with a large number of cloud application workloads.

Acknowledgments

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 104-2221-E-029-010-MY3 and MOST 106-3114-E-029-003. This work was also financially supported by the Intelligence Recognition Industry Service Research Center (AIR-IS Research Center) from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

References

[1] A. Hakiria, A. Gokhale, P. Berthou, D.C. Schmidt, T. Gayraud, Software-defined networking: Challenges and research opportunities for future internet, Comput. Netw. 75 (2014) 453–471.
[2] F. Rodrguez-Haro, F. Freitag, L. Navarro, E. Hernnchez-snchez, N. Faras-Mendoza, J.A. Guerrero-Ibez, A. Gonzlez-Potes, A summary of virtualization techniques, Procedia Technol. 3 (2012) 267–272.
[3] R.S. Couto, M.E.M. Campista, L.H.M. Costa, Network resource control for xen-based virtualized software routers, Comput. Netw. 64 (2014) 71–88.
[4] S. Huang, J. Griffioen, K.L. Calvert, Network hypervisors: Enhancing sdn infrastructure, Comput. Commun. 46 (2014) 87–96.
[5] T. Voith, K. Oberle, M. Stein, Software defined networking for security enhancement in wireless mobile networks, Comput. Netw. 66 (2014) 94–101.
[6] P. Qin, B. Dai, B. Huang, G. Xu, Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data, CoRR abs/1403.2800 (2014).
[7] M.H. Raza, S.C. Sivakumar, A. Nafarieh, B. Robertson, A comparison of software defined network (sdn) implementation strategies, in: ANT/SEIT, pp. 1050–1055.
[8] G. Yi, S. Lee, Fully distributed handover based on sdn in heterogeneous wireless networks, in: ICUIMC, p. 70.
[9] P. Smith, A.E.S. Filho, D. Hutchison, A. Mauthe, Management patterns: Sdn-enabled network resilience management, in: NOMS, pp. 1–9.
[10] A.F. Trajano, M.P. Fernandez, Two-phase load balancing of in-memory key–value storages using network functions virtualization (nfv), J. Netw. Comput. Appl. 69 (2016) 1–13.
[11] S. Ayoubi, C. Assi, Y. Chen, T. Khalifa, K.B. Shaban, Restoration methods for cloud multicast virtual networks, 78 (2017) 180–190.
[12] H. Yang, L. Cheng, J. Yuan, J. Zhang, Y. Zhao, Y. Lee, Multipath protection for data center services in openflow-based software defined elastic optical networks, Opt. Fiber Technol., Mater. Devices Syst. (2015).
[13] J. Matias, A. Mendiola, N. Toledo, B. Tornero, E. Jacob, The ehu-oef: An openflow-based layer-2 experimental facility, Comput. Netw. 63 (2014) 101–127.
[14] Rashi Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, R. Boutaba, Network Function Virtualization: State-of-the-Art and Research Challenges, Vol. 18, IEEE, 2016, pp. 236–262.
[15] J.M.A. Calero, J.G. Aguado, Comparative analysis of architectures for monitoring cloud computing infrastructures, Future Gener. Comput. Syst. 47 (2015) 16–30.
[16] A. Corradi, M. Fanelli, L. Foschini, Vm consolidation: A real case based on openstack cloud, Future Gener. Comput. Syst. 32 (2014) 118–127.
[17] O. Litvinski, A. Gherbi, Experimental evaluation of openstack compute scheduler, Procedia Comput. Sci. 19 (2013) 116–123.
[18] Z. Li, H. Li, X. Wang, K. Li, A generic cloud platform for engineering optimization based on openstack, Adv. Eng. Softw. 75 (2014) 42–57.
[19] Netflow, http://en.wikipedia.org/wiki/NetFlow, (1996).
[20] Openvswitch, http://openvswitch.org/, [2014].
[21] J. Pettit, J. Gross, B. Pfaff, M. Casado, Virtual Switching in an Era of Advanced Edges, Data Center - Converged and Virtual Ethernet Switching, 2010.
[22] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, Extending Networking into the Virtualization Layer, Hot Topics in Networks, 2009.
[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, Comput. Commun. Rev. 38 (2008) 69–74.
[24] Opendaylight, http://www.opendaylight.org/, 2018.
[25] Openstack, https://www.openstack.org/, 2018.
[26] Neutron, https://wiki.openstack.org/wiki/Neutron/, 2018.
[27] The Standard Normal Distribution, The Standard Normal Distribution http://www.oswego.edu/~srp/stats/z.htm/, 2018.

Chao-Tung Yang received a B.Sc. degree in Computer Science from Tunghai University, Taichung, Taiwan, in 1990, and the M.Sc. degree in Computer Science from National Chiao Tung University, Hsinchu, Taiwan, in 1992. He received the Ph.D. degree in Computer Science from National Chiao Tung University in July 1996. In August 2001, he joined the Faculty of the Department of Computer Science at Tunghai University as an Associate Professor. He is a full Professor started in August 2007 and as a Distinguished Professor in August 2015. He is serving in a number of journal editorial boards, including Future Generation Computer Systems, International Journal of Communication Systems, KSII Transactions on Internet and Information Systems, Journal of Cloud Computing, IJ-CLOSER, International Journal of Next-Generation Computing (IJNGC), Dr. Yang has published more than 300 papers in journals, book chapters and conference proceedings. His present research interests are in Cloud computing and Big data, Parallel and multicore computing, and Web-based applications. He is both a member of the IEEE Computer Society and ACM. He is also both a member of IICM and TACC in Taiwan.

Shuo-Tsung Chen received the B.S. degree in Mathematics from National Cheng Kung University, Tainan in 1996 and M.S. degree in Applied Mathematics from Tunghai University, Taichung in 2003, Taiwan. In 2010, he received the Ph.D. degree in Electrical Engineering from National Chinan University, Nantou, Taiwan. Now he is an Assistant Professor in National Yunlin University of Science and Technology, Taiwan.

Jung-Chun Liu received his B.S. degree in Electrical Engineering from National Taiwan University in 1990. He received his M.S. and Ph.D. degrees from the Department of Electrical and Computer Engineering at the University of Texas at Austin, in 1996 and 2004, respectively. He is currently an Assistant Professor in the Department of Computer Science at the Tunghai University, Taiwan. His research interests include cloud computing, embedded systems, big data, network security, artificial intelligence, and wireless sensor networks.

Yao-Yu Yang received the B.S. degree in 2013 and M.S. degree in 2015 in Computer Science from Tunghai University, Taichung, Taiwan. Now he is a Computer Engineer in inwinSTACK Co., Ltd.

Karan Mitra is an Assistant Professor at Luleå University of Technology, Sweden. He received his Dual-badge Ph.D. from Monash University, Australia and Luleå University of Technology in 2013. He received his MIT (MT) and a PGradDipDigComm from Monash University in 2008 and 2006, respectively. He received his BIS (Hons.) from Guru Gobind Singh Indraprastha University, Delhi, India in 2004. His research interests include quality of experience modeling and prediction, context-aware computing, cloud computing and mobile and pervasive computing systems. He is a member of the IEEE and ACM.

Dr. Rajiv Ranjan is a Reader in the School of Computing Science at Newcastle University, UK; chair professor in the School of Computer, Chinese University of Geosciences, Wuhan, China; and a visiting scientist at Data61, CSIRO, Australia. His research interests include grid computing, peer-to-peer networks, cloud computing, Internet of Things, and big data analytics. He has published about 200 research papers (including 120+ journal papers). His papers have received 7770+ Google Scholar citations in total, he has an h-index and i10-index of 36 and 74 respectively. His papers have also received 1700+ citations and h-index of 16; according to Thomson Reuters Journal Citation Report (goo.gl/mJVphW). He also has an Scopus (Author ID:22980683700) h-index of 19 and total citations >2900. Ranjan has a Ph.D. in Computer Science and Software Engineering from the University of Melbourne (2009). Contact him at raj.ranjan@ncl.ac.uk or http://rajivranjan.net.