DESIGN, IMPLEMENTATION, AND OPERATION OF IPv6-ONLY IaaS SYSTEM WITH IPv4-IPv6 TRANSLATOR FOR TRANSITION TOWARD THE FUTURE INTERNET DATACENTER

Keiichi Shima[1], Wataru Ishida[2] and Yuji Sekiya[3]

[1]*IIJ Innovation Institute, 1-105 Kanda-jinbocho, Chiyoda-ku, Tokyo, Japan*

[2]*The University of Tokyo, 3-8-1 Komaba, Meguro-ku, Tokyo, Japan*

[3]*The University of Tokyo, 2-11-16 Yayoi, Bunkyo-ku, Tokyo, Japan*

keiichi@iijlab.net, wataru@hongo.wide.ad.jp, sekiya@wide.ad.jp

Keywords: Cloud Middleware Frameworks, IaaS, Scalability, Redundancy, Cost Reduction, IPv6, IPv4 Compatibility.

Abstract: Internet operation is migrating from IPv4-only operation to IPv4/IPv6 dual-stack operation. From the viewpoint of the operation cost, it is best if the operation can be done only with a single protocol, either IPv4 or IPv6. However, it is mandatory for service providers to accept all three types of users, 1) IPv4-only users who will remain in the Internet for long time, 2) IPv4/IPv6 dual-stack users who will be a dominant users in near future, and 3) IPv6-only users who will appear in the future as IPv6 deployment progresses. In this paper, we propose a recommended operation model designed for the IaaS system operated with IPv6-only network with a wide area L2 network and IPv4-IPv6 translation software for backward compatibility. With our proposal, we can reduce the use of IPv4 addresses in the cloud backend network, and provide high-performance, scalable, and redundant address translation software suitable for the IPv6-only IaaS system that can be one of the reference operation models of future datacenters.

1 INTRODUCTION

The computing power of computers is growing every year. However, many studies show that there is an obvious limitation of computing power if we stick one single processor system. Because of this reason, cloud computing technologies are getting attention of many people. Researchers are now investigating its usage, application model, and operation techniques to utilize the limited resources more effectively. At the same time, virtualization technologies that enable us to slice one physical computer into several virtual computer resources to suppress idle resources as much as possible. With the virtualization technologies, it is expected that we will have a huge computer network that includes far larger number of virtual computers than the physical computers.

The network protocol used to interconnect computers has been the IPv4 protocol for long time. However, because of recent depletion of IPv4 addresses, the successor protocol, IPv6, is now being deployed. Since we can still use IPv4 addresses and its network even though we do not have new IPv4 addresses, the future network will be a mixed network of IPv4 and IPv6 for many years until IPv4 disappears completely.

Service providers must support both IPv4 and IPv6, because service users will be mixed users of IPv4-only users, IPv4/IPv6 dual-stack users, and IPv6-only users in the future. Although IPv4 and IPv6 are technically quite similar except the size of their address space, however, these protocols do not have interoperability because their protocol header formats are completely different. The operators have to manage two similar networks. That is a duplicated effort.

If we can manage and operate a single stack network for the service backend nodes and place dual-stack nodes only as the entry points to the services, then we can reduce the operation cost to manage two duplicated networks. We can also quickly follow the technology advance of the network by focusing on only one single protocol stack. In this paper, we focus on a virtual computer resource management environment (IaaS, Infrastructure As A Service) and propose a system whose backend network is operated by IPv6-only with dual-stack public nodes as service entry points using IPv4-IPv6 protocol translator software. This kind of IPv4-IPv6 protocol translator is not a new technology. It was proposed almost as same time as IPv6 was proposed. However, such a translator is basically assuming to map one IPv4 address to multiple IPv6 addresses, because its main usage is to provide IPv4 connectivity to IPv6 only nodes originally. This means the software tends to become complicated because it has to keep track on the address mapping status and session status among multiple IPv6 addresses. It also has a scalability and redundancy problem because it needs to manage many mapping information and session status and this status information is hard to share and synchronize among multiple translator nodes. Our proposed system particularly insists on one-to-one address mapping which doesn't have problems addressed above, with the assumption that the translator is used as a part of an IaaS system. With this compromise, we can design a realistic IaaS system without having problems of existing IPv4-IPv6 translation system as described before and realize IPv6-only backend service operation.

2 RELATED WORKS

2.1 IaaS Model

Needless to say, most of the current service providers are using IPv4 as their base IP protocol to build services. As IPv6 is getting popular, some of them are now considering to supporting dual-stack service using translators. Although there are several transition scenarios to move from IPv4 to IPv6 (Mackay et al., 2003), they basically use IPv4 for server nodes and provide IPv6 connectivity to IPv6 clients using translator technology. This is the biggest difference from our approach. We use IPv6 addresses for server nodes and provide dual stack connectivity to client nodes. Considering that the IPv6 clients are going to grow in the future, focusing on IPv6 and providing IPv4 as an additional service seems the right choice.

The traditional translation operation is shown in section 2.1 of Chen's paper (Chen, 2011). We think that the model has several drawbacks. First, since it uses IPv4 for their backend service networks, it may face the IPv4 depletion problem. Second, if they use private IPv4 address space to avoid the IPv4 depletion issue, they may face IPv4 NAT traversal issues for servers they are operating in their private networks. Third, if they use NAT for their servers, they have to operate two NAT services, one for IPv4 private addresses, and the other for IPv6-IPv4 addresses. In our proposed model, IPv6 services are provided natively, and IPv4 services are provided using translation. Detailed discussion is provided in section 3.1.

2.2 Translators

Discussion to develop technologies to bridge IPv4 and IPv6 was started as early as when IPv6 protocol designers decided not to provide IPv4 backward compatibility in IPv6. Waddington and Chang summarized IPv4-IPv6 transition mechanisms in their paper (Waddington and Chang, 2002). From the viewpoint of translation mechanisms, we can classify these transition technologies into three approaches. The first approach is providing interoperability in the application layer, the second approach is relaying data in the transport layer, and the last is converting an IP header in the IP layer.

The first approach, the application layer approach is performed by proxy server software designed for each application protocol that needs interoperability between IPv4 and IPv6. The proxy server will be located at the boundary of IPv4 and IPv6 networks and accept specific application requests (e.g. web requests) from client nodes. The proxy server interprets the request contents, build a new request for the destination server, and send it using a proper IP protocol. The response from the server node is intercepted and resent by the proxy server as well. Since the proxy server understands the application protocol completely, this approach can be most flexible compared to the other approaches. On the other hand, development per application protocol is required which will increase development and operation cost. This approach is mostly used when we only need complex context processing in between IPv4 and IPv6 networks, since many services can be interoperable using the rest of the approaches.

In the second approach, the transport layer approach, the relay server located in between the IPv4 and IPv6 networks terminates incoming connections from clients at the socket level, and starts new transport connections to the destination server on the other side of the network. The well-known mechanisms of this approach are, for example, SOCKS64 (Kitamura, 1999; Kitamura, 2001) and Transport Relay Translator (itojun Hagino and Yamamoto, 2001). In SOCKS64, the client side SOCKS64 library replaces the DNS name resolution and socket I/O functions to intercept all the client traffic and forward to a SOCKS64 server. This mechanism requires client applications to be updated to use the SOCKS64 library, however, once the library is replaced, applications do not need to pay any attention to IPv4-IPv6 translation issues. The Transport Relay Translator does not require any update to client applications. Instead, the mechanism assumes that clients use specially crafted IPv6 addresses in which IPv4 address of the server is embedded, when IPv6 clients are going to connect to IPv4 servers. The relay server terminates the incoming IPv6 connections from client nodes and starts new connections to the destination servers using the server address extracted from the special IPv6 address. Such special IPv6 addresses are usually provided by the site level special DNS server in the client site that supports DNS64 (Bagnulo et al., 2011b).

The third approach, the IP layer header translation is similar to the NAT technology used in IPv4. Examples of this approach are NAT-PT (Tsirtsis and Srisuresh, 2000) and NAT64 (Bagnulo et al., 2011a). In this approach, the translation server does not need to terminate connections between clients and servers. The address fields and other header fields of packets reached to the translator node will be translated based on the predefined header transformation rules between IPv4 and IPv6.

These related technologies were originally designed for the situation that there are IPv4 server nodes in the Internet, and IPv6 client nodes are going to connect to those IPv4 servers. In a real operation, it is assumed that one or a few IPv4 addresses are allocated to the translation server, and many IPv6 client nodes share the IPv4 addresses. However, the system discussed in this paper assumes the opposite case, that is, there are IPv6 server nodes in a cloud system, and existing IPv4 or IPv6 client nodes are going to connect to the IPv6 servers. In this case, we cannot share IPv4 addresses between many IPv6 servers without any modification to the IPv4 client side. Because of this nature, we have to map one IPv4 address to one IPv6 server eventually. Of course, the previous works and proposed mechanisms can do the same scenario in theory, however, there are few implementations that can support the case we are assuming.

There are some evaluation studies in these translation technologies (Mackay and Edwards, 2006; Škoberne and Ciglarič, 2011). However, since the performance of the mechanisms highly depends on how the mechanisms are implemented, we need to compare whenever we design a new translation software.

There are several commercial devices providing IPv4-IPv6 translation function. Most of them are targeting a translation service of internal IPv6 client connection requests generated inside the customer's network, to global IPv4 servers. Different from the existing translation products, our approach targets on service provider side and provides a function for IPv4 legacy clients to connect servers running with IPv6 only.

Our approach runs without keeping any state information of connections, while most of translator products are stateful. This is because existing translator products are for IPv6 client nodes, and shares one IPv4 address among several IPv6 clients. Our proposal assumes completely different operation, that is, translation service for servers. Because of this difference, our approach is implemented as stateless service. That makes it easier to place multiple translators at several network exit points and distribute traffic load.

In this paper, we propose a new IaaS system design and IPv4-IPv6 translator design and implementation that only use IPv6 as its service backend network, providing dual-stack service to both IPv4 and IPv6 clients. We also give the performance measurement results of the system.

3 DESIGN AND IMPLEMENTATION OF THE IaaS SYSTEM AND THE TRANSLATION SOFTWARE

In this section, we describe the overview of the proposed IaaS system and design of the IPv4-IPv6 header translation software.

3.1 Design and Implementation of the IaaS System

The system we are targeting is an IaaS system that provides virtual computers as a unit of resources using virtualization technology. Service integrators can use virtual computers as their service components by requesting the IaaS system to slice physical computers to make virtual computers whenever needed. The actual configuration of a service varies in each service, however, one service usually composed of several computers, such as a web frontend node, load balancers, a database node, and so on. Services such as a distributed storage may need larger number of virtual computers as a backend system. When we consider providing a dual-stack service to users, one possible implementation is building the IaaS system as a complete dual-stack system. However, as we discussed in section 1, we will have a duplicated effort to maintain two different network stacks, which they have almost same functions. Since the service users usually only see the frontend nodes, it will be sufficient if we can make those frontend nodes dual-stack. The communication between the frontend nodes and backend nodes do not necessarily be dual-stack. By focusing only one protocol stack inside the IaaS system, the network

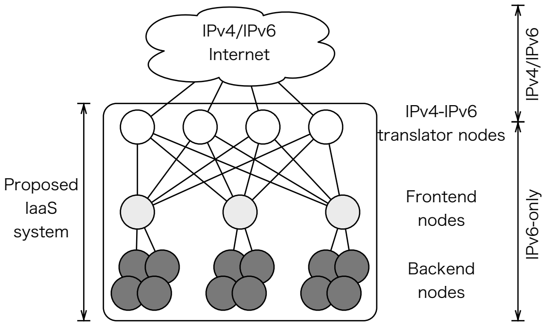

Figure 1: The overview of the proposed IaaS system.

design will be simpler that will benefit the IaaS service provider, and the service development cost and test cost can also be reduced that will benefit service providers.

In this proposed system, we assume an IPv4 address and an IPv6 address are mapped one to one. Because the proposed system discussed in this paper is for IaaS providers providing server resources to their users, who are PaaS providers, SaaS providers, and/or ASPs, we cannot design the system to share one IPv4 address by several IPv6 server nodes. We also don't consider implementing this requirement using application level proxy solution, since service providers have different requirements for their services, we cannot limit the communication layer function to application layers. This may be a problem considering that the IPv4 address is scarce resource. However, we need to use IPv4 address anyway if we want to support IPv4 clients. And also we need to map addresses one to one if we want to provide services without any modification on the client side. Our best effort to handle this issue is that we do not allocate IPv4 addresses to backend nodes to avoid non-necessary IPv4 use.

The version of an IP protocol used in the backend network can be either IPv4 or IPv6. In the implementation shown in this paper, we use IPv6 as an internal protocol for the backend network considering the trend of IPv6 transition of the whole Internet.

Figure 1 depicts the overview of the proposed system. We said that the frontend nodes provide dual-stack services to the client before, however precisely speaking, these frontend nodes do not have any IPv4 addresses. The mapping information between IPv4 and IPv6 addresses are registered to each IPv4-IPv6 translator node and shared by all the translator nodes. Since the mapping is done in the one to one manner, no translator nodes need to keep session information of ongoing communication. They can just translate IP packets one by one. This makes it possible to place multiple translator nodes easily to scale out the translation service when the amount of the traffic grows. This feature also contributes robustness of the translation system. When one of the translator nodes fails, we can just remove the failed node from the system. Since there is no shared session information among translator nodes, the service is kept by the rest of the translator nodes without any recovery operation.

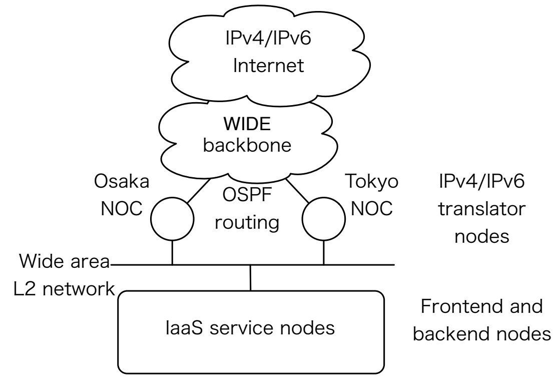

Figure 2: The actual network configuration of the proposed IaaS system implemented and operated in the WIDE project operation network.

Figure 2 shows the overview of the actual system we are operating. The virtual machine resource management system located at the bottom of the figure is the WIDE Cloud Controller (WCC) (WIDE project, 2011) developed by the WIDE project. The two network operation centers (NOC) located at Tokyo and Osaka are connected by a wide area layer 2 network technology. The layer 2 network is a 10Gbps fiber line dedicated to the WIDE project network operation. There are two layer 3 networks in Tokyo and Osaka NOC whose network address spaces are same. These two layer 3 networks are connected by the layer 2 link using VLAN technology. The translator nodes are placed at each location. The routing information of the IPv4 addresses used to provide IPv4 connectivity to IPv4 clients is managed using the OSPFv3 routing protocol in the WIDE project backbone network. Since all the translator nodes advertise the same IPv4 address information using the equal cost strategy, incoming traffic is distributed based on the entry points of the incoming connections. The aggregated IPv4 routing information is advertised to the Internet using the BGP routing protocol. Any incoming IPv4 connection requests from the Internet are routed to the WIDE backbone based on the BGP information, routed to one of the translator nodes based on the OSPFv3 routing information, and finally routed to the corresponding virtual machine based on the static mapping table information stored in the translator nodes. The translation mechanism is described in section 3.2. Failure of either Osaka or Tokyo NOC will result in failure of the OSPFv3 routing infor-

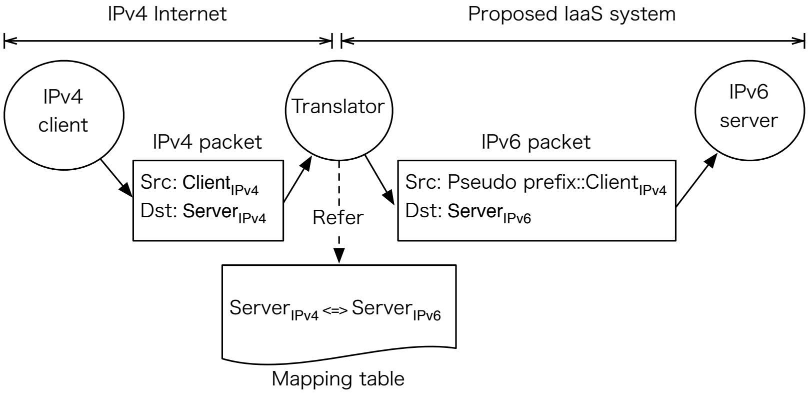

Figure 3: IPv4-IPv6 header translation procedure.

mation advertisement from the failed NOC, however, thanks to the nature of a routing protocol, and one to one stateless IPv4-IPv6 mapping mechanism, the other NOC and translator will continue serving the same address space to the Internet.

3.2 Design and Implementation of the Translator Software

The server nodes in the IaaS backend system will see all the communication from their clients as IPv6 connections, since the backend system is operated in IPv6 only. All the IPv4 traffic from clients are intercepted by the translator nodes and converted to IPv6 traffic. Figure 3 shows the translation procedure used in the translator software.

IPv4 address information bound to IPv6 addresses of servers is stored in the mapping table. Based on the table, the destination address of the IPv4 packet from client nodes is converted to the proper IPv6 server address. The source address of the IPv6 packet is converted to a pseudo IPv6 address by embedding the IPv4 address of the client node to the lower 32-bit of the pseudo IPv6 address. The pseudo IPv6 address is routed to the translator node inside the backend network. The reply traffic from the server side is also converted with the similar but opposite procedures.

The mechanism has been implemented using the tun pseudo network interface device that is now provided as a part of the basic function of Linux and BSD operating systems, originally developed by Maxim Krasnyansky[1]. The tun network interface can capture incoming IP packets and redirect them to a user space program. The user space program can also send IP packets directly to the network by writing the raw data of the IP packets to the tun interface. The translator nodes advertise the routing information of the IPv4 addresses in the mapping table to receive all the traffic sent to those IPv4 addresses. The incoming packets are received via the tun network interface and passed to the user space translator application. The translator performs the procedure described in figure 3 to translate the IPv4 packet to IPv6 packet. The converted IPv6 packet is sent back to the tun network interface and forwarded to the real server that is running with IPv6 address only.

For the reverse direction, the similar procedure is applied. Since the translator nodes advertise the routing information of the pseudo IPv6 address that includes the IPv4 address of the client node is advertised to the IaaS backend network, the translator nodes receive all the reverse traffic. The nodes convert those IPv6 packets into IPv4 packets using the embedded IPv4 client address and mapping table information, and forward to the original IPv4 client.

This translator software is published as map646 open source software (Keiichi Shima and Wataru Ishida, 2011), and anyone can use it freely.

4 SYSTEM EVALUATION

We implemented the proposed IaaS system as a part of the WIDE project service network as shown in figure 2. By focusing on IPv6-only operation, we could be free from IPv4 network management.

For the redundancy, we located two translator nodes in different locations of the WIDE project core network. We sometimes stop one of them for maintenance. In that case, the other running node is working as a backup node. We confirmed that the redundancy mechanism is working automatically in a real operation.

The incoming traffic to IPv4 addresses are load-balanced based on the BGP and OSPFv3 informa-

[1] http://vtun.sourceforge.net/tun/

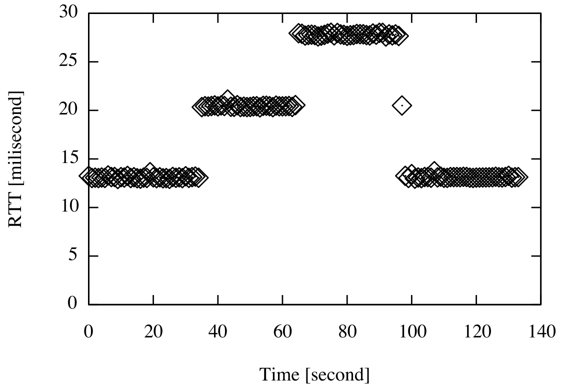

Figure 4: RTT measurement in case of failure of a translator.

tion. For the outgoing traffic, currently a simple router preference mechanism is used to choose an upstream router from IPv6 servers. We are considering using more advanced traffic engineering methods, such as destination prefix based routing in near future.

Figure 4 shows the RTT measurement result from an IPv4 client to an IPv6 server. Initially, both translator nodes are running. At time 35, we stopped the router advertisement function of translator in Tokyo. The traffic from outside to the cloud network still goes to Tokyo node, but the returning traffic will be routed to Osaka. At around time 65, we stopped routing function of the Tokyo node. After this point, all the traffic goes to Osaka and returns from Osaka. We restarted the routing function of Tokyo node, and restarted router advertisement at Tokyo node at around time 90. Finally, all the traffic came back to go through Tokyo node.

5 PERFORMANCE EVALUATION

The obvious bottleneck of the system is the translator nodes where all the traffic must go through with them. This section shows the evaluation result of the translation software.

5.1 Translation Performance

The performance evaluation is done with the four different configurations shown in figure 5. The configuration 1 and 2 (C1 and C2) are the translation cases using our translator software. Configuration 2 (C2) and 3 (C3) use normal IPv4 and IPv6 forwarding mechanisms to compare the translation performance with no translation cases.

Evaluation is done using two methods, one is the ping program to measure RTT, and the other is the iperf program to measure bandwidth. All the results in this experiment show the average value of 5 measurement tries[2]. The computer nodes used in this performance test are shown in table 1, and all the tests were performed locally by directly connecting 3 computers as shown in figure 5.

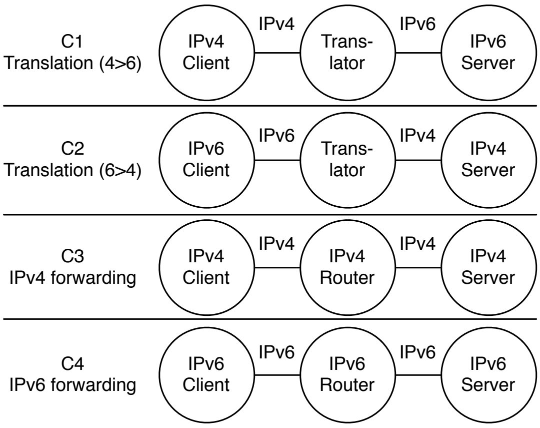

Figure 5: The four configurations used in performance evaluation.

Table 1: Specification of nodes.

	Client/Server	Translator/Router
CPU	Core2 Duo 3.16GHz	Xeon L5630 2.13GHz × 2
Memory	4GB	24GB
OS	Linux 3.0.0-12-server	Linux 3.0.0-12-server
NIC	Intel 82573L	Intel 82574L

Figure 6 shows the result of the ping test. The RTT values were 0.45ms in C1, 0.43ms in C2, 0.36ms in C3, and 0.36ms in C4.

Figure 7 shows the result of the TCP bandwidth measurement test. The bandwidth values were 923.8Mbps in C1, 922.8Mbps in C2, 938.0Mbps in C3, and 925.8Mbps in C4.

Figure 8 shows the result of the UDP bandwidth measurement test. We changed the transmission rate of the sender side from 100Mbps to 1000Mbps with 100Mbps step, and measured how much bandwidth was achieved at the receiver side. The maximum bandwidth values were 786.0Mbps in C1, 802.0Mbps in C2, 802.0Mbps in C3, and 802.0Mbps in C4.

[2] We didn't record standard deviation of these tries, since the results were stable.

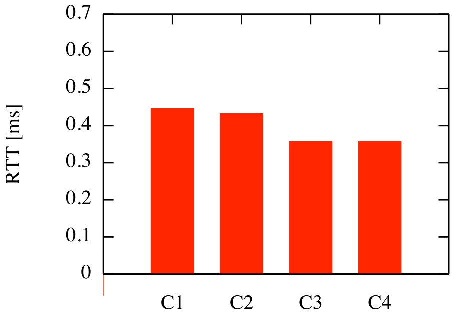

Figure 6: RTT measurement result using the ping program.

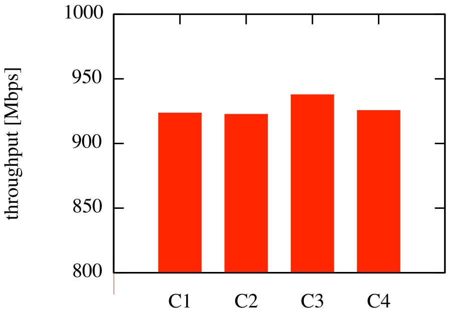

Figure 7: TCP bandwidth measurement using the iperf program.

5.2 Comparison with Related Method

In this section, we compare the translation performance of map646 to linuxnat64 (Julius Kriukas, 2012) which is one of the NAT64 implementations. The main usage of NAT64 is to provide access to IPv4 servers from IPv6-only nodes. Because of the difference of the usage scenario, we located a server in the IPv4 network side, and located a client in the IPv6 network side in this test (the C2 case). This is opposite to our proposed IaaS usage, however we think the test can give us meaningful result in the sense of performance comparison. In this test, we used Linux 2.6.32 instead of 3.0.0-12-server because linuxnat64 did not support Linux version 3 when we performed this test. There is no big difference of IPv4/IPv6 forwarding performance between Linux version 2 and 3. We are planning to test again with a newer kernel once linuxnat64 supports it.

Figure 9 shows the RTT measurement result. The values are 0.55ms in the map646 case, and 0.39ms in the linuxnat64 case.

Figure 10 shows the TCP bandwidth measurement result. The bandwidth values were 903.2Mbps in the map646 case, and 879.8Mbps in the linuxnat64 case. Figure 11 is the result of the UDP throughput measurement. The maximum bandwidth values were 943.0Mbps in the map646 case, and 943.0Mbps in the

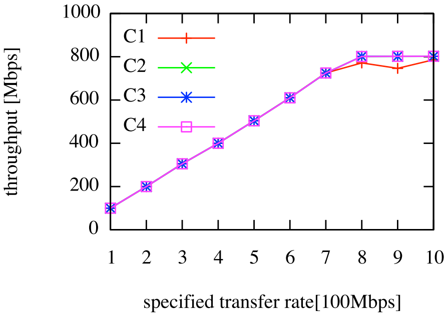

Figure 8: UDP bandwidth measurement using the iperf program.

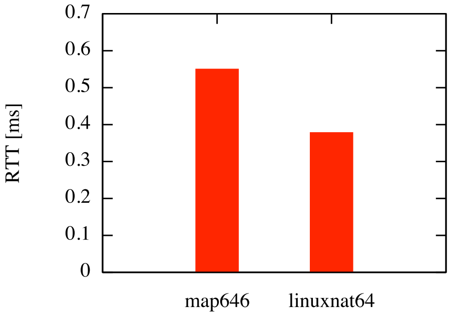

Figure 9: RTT comparison of map646 and linuxnat64.

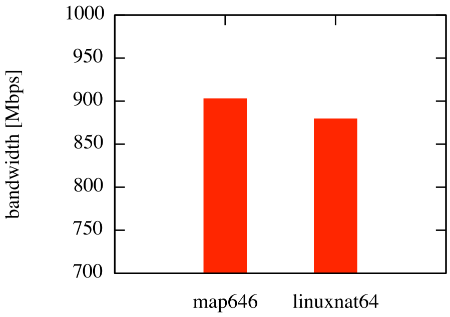

Figure 10: TCP bandwidth comparison of map646 and linuxnat64.

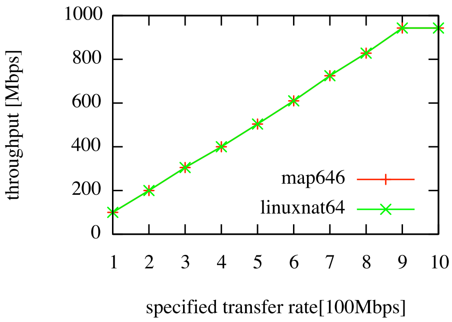

Figure 11: UDP bandwidth comparison of map646 and linuxnat64.

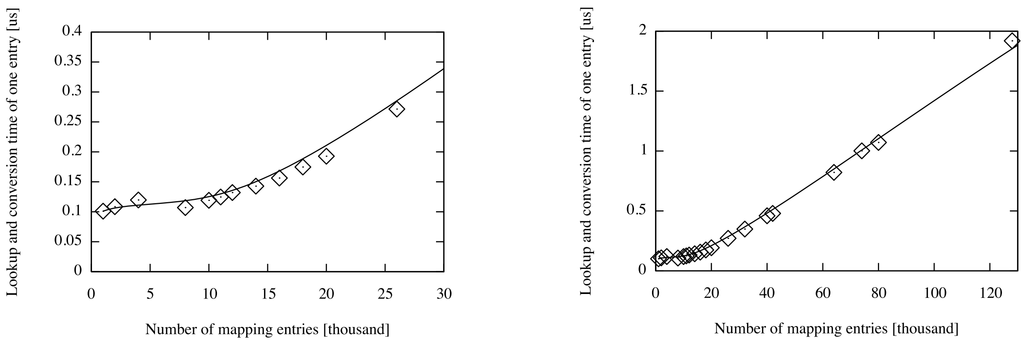

Figure 12: Simulated result of mapping table lookup and address conversion overhead (left figure: 1000 to 26000 entries, right figure: 1000 to 128000 entries).

linuxnat64 case.

5.3 Consideration

Based on the observation in section 4, we could consider the proposed IaaS system could reduce the maintenance cost, achieve redundancy and scalability.

For the performance, as shown in figure 6, the RTT degradation is around 0.07ms to 0.09ms. This is reasonable because map646 is implemented as a user space program while IPv4 and IPv6 forwarding are implemented inside kernel. The other potential reason of the degradation is a mapping entry lookup overhead. In this experiment, we only defined two mapping entries in map646. We have not measured the overhead when there are a lot of mapping entries, however we think it will not affect the total performance because we are using a hash table for mapping information lookup. Figure 12 shows the simulated result of the lookup and address conversion overhead for different number of mapping entries. In our implementation, the size of the hash table is 1009. From the left figure in figure 12, we can see the table lookup and address conversion is done in almost constant time when the size of the mapping entries is less than 10000. When the size of the table grows, the lookup and conversion time also grows linearly as can see in the right figure in figure 12. The lookup and conversion overhead of one entry is around 2 μs when there are 128 thousands of mapping entries. This value is enough small that can be ignored compared to the RTT degradation value. However we agree that the real measurement of the performance degradation with many mapping entries is an important topic, and it is one of our future works.

The forwarding performance of map646 is 1.5% to 1.6% worse than the normal forwarding case in the TCP case, and 2.0% in the UDP case. We actually did not see a big degradation in both TCP and UDP cases. This means that the translation itself is enough fast to process all the incoming packets. We can conclude that the performance of the map646 translator software is acceptable for the real operation. One thing we have to note is that the test is done using 1Gbps network interfaces. Because of recent advance of processor technology, 1Gbps is not too fast any more. We will perform further measurement in faster environment such as 10Gbps network interfaces.

From figure 10 and 11, we can see almost no degradation compared to linuxnat64 implementation. Map646 achieved even better performance than linuxnat64 in the TCP case. We expected a slight degradation in the map646 case, since linuxnat64 is implemented as a kernel module while map646 is a user space program, but the result did not show any big difference. This result shows that the performance highly depends on how the software is implemented. As similar as the previous result, we will perform the same test using 10Gbps networks to verify the performance of map646 in a broader bandwidth environment as follow-up works.

We are actually operating several web servers as a part of our daily operation. The examples of those servers are the WIDE project web server[3], the web server of the Graduate School of Information Science and Technology at the University of Tokyo Japan[4], the DNS server for the wide.ad.jp domain, the web server for the Internet Conference[5], the KAME project web server[6], and so on. The real operation of these servers also proves the system usability.

Finally we note some of the concerns of the proposed system. This proposed system requires the

[3] http://www.wide.ad.jp/

[4] http://www.i.u-tokyo.ac.jp/

[5] http://www.internetconference.org/

[6] http://www.kame.net/

same number of IPv4 addresses as the frontend server nodes. This is a trade-off between backward compatibility and IPv4 address usage efficiency. When IPv4 becomes a minor protocol, then probably more efficient IPv4 address sharing mechanisms for server nodes may be deployed. The other concern is that since the proposed mechanism translates addresses, server nodes may require additional security considerations such as filtering. When writing down filter rules, the server operators need to pay attention to the pseudo IPv6 address space that covers the entire IPv4 address space.

6 CONCLUSIONS

In this paper, we proposed a new operation model for IaaS service providers to adapt the future IPv6 Internet. In the proposed system, we suggest the IaaS service providers should focus on a single stack operation as their backend network system. That will decrease the operation cost compared to the dual-stack operation style. Considering recent trend of IP technology, we think IPv6 is a better choice for the backend network. We also designed a robust and scalable IPv4-IPv6 protocol translator software and implemented it. The measurement result showed the software has a slight degradation compared to the native forwarding cases, however, the performance was acceptable. We deployed the idea in our real operation network. We are actually operating the IaaS system and several real web servers as our daily service infrastructure at this moment. With all these results and observation, we conclude that the proposed IaaS system is useful and feasible as one of the future IaaS operation models.

REFERENCES

Bagnulo, M., Matthews, P., and van Beijnum, I. (2011a). *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*. IETF. RFC6146.

Bagnulo, M., Sullivan, A., Matthews, P., and van Beijnum, I. (2011b). *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers*. IETF. RFC6147.

Chen, G. (2011). *NAT64 Operational Considerations*. IETF. draft-chen-v6ops-nat64-cpe-03.

Itojun Hagino, J. and Yamamoto, K. (2001). *An IPv6-to-IPv4 Transport Relay Translator*. IETF. RFC3142.

Kriukas, J. (2012). Linux NAT64 implementation. http://sourceforge.net/projects/linuxnat64/.

Shima, K. and Ishida, W. (2011). map646: Mapping between IPv6 and IPv4 and vice versa. https://github.com/keiichishima/map646/.

Kitamura, H. (1999). Entering the IPv6 Communication World by the SOCKS-Based IPv6/IPv4 Translator. In *INET99*.

Kitamura, H. (2001). *SOCKS-based IPv6/IPv4 Gateway Mechanism*. IETF. RFC3089.

Mackay, M. and Edwards, C. (2006). A Comparative Performance Study of IPv6 Transitioning Mechanisms NAT-PT vs. TRT vs. DSTM. In *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, volume 3976 of *Lecture Notes in Computer Science*, pages 1125–1131. Springer Berlin / Heidelberg.

Mackay, M., Edwards, C., and Dunmore, M. (2003). A Scenario-Based Review of IPv6 Transition Tools. *IEEE Internet Computing*, 7(3):27–35.

Tsirtsis, G. and Srisuresh, P. (2000). *Network Address Translation - Protocol Translation (NAT-PT)*. IETF. RFC2766.

Škoberne, N. and Ciglarič, M. (2011). Practical Evaluation of Stateful NAT64/DNS64 Transition. *Advances in Electrical and Computer Engineering*, 11(3):49–54.

Waddington, D. G. and Chang, F. (2002). Realizing the transition to ipv6. *IEEE Communications Magazine*, 40(6):138–147.

WIDE Project (2011). WIDE Cloud Controller. http://wcc.wide.ad.jp/.