A Tunnels-Between-Hosts based Method for IPv4 Application over Native IPv6 Network

Xiangbin Cheng, Jun Bi, Xing Li and Jianping Wu
Network Research Center, Tsinghua University, Beijing, 100084, China
{cxb,junbi}@netarchlab.tsinghua.edu.cn

Abstract

Along with the development of IPv6, native IPv6 networks will begin to appear. It would be helpful if we can provide an effective mechanism for existing IPv4-only applications to work in those native IPv6 networks. In this paper, we analyze the problems of some current transition methods, and present a new mechanism based on Tunnels-Between-Hosts. Our strategy is to provide the IPv4-only application with a temporary IPv4 "virtual address", which is negotiated directly between two communicating hosts using IPv6. We give a detailed description of our mechanism and compare it with some other mechanisms.

1. Introduction

In order to solve the problems of IPv4 protocol and find a better protocol for the future, IETF proposed IPv6 for the next-generation Internet. During the transition, both network infrastructure and applications need to be upgraded. However, many applications using IPv4 will not be able to work for IPv6 due to format differences between IPv4 and IPv6 addresses. Updating current applications to support IPv6 is one solution, but it will consume a lot of time. IPv6 transition facilitates native IPv6 network. Thus, it would be helpful to provide a mechanism for IPv4 applications to work in native IPv6 environment.

In this paper, we present a transition mechanism based on Tunnels-Between-Hosts. Our strategy is to provide a temporary IPv4 "virtual address" for IPv4 applications in the upper layer by inserting modules in the host's protocol stack. The temporary IPv4 address is negotiated directly between two communicating IPv6 hosts through an effective protocol, and data packets are encapsulated in IPv6 headers and sent through IPv4-in-IPv6 tunnels between hosts. In Section 2, we analyze the problem. In Section 3, some of other existing mechanisms are examined and followed by the study in a given scenario. We describe our mechanism and compare it with related work in Section 4, and summarize our work in Section 5.

2. Problem Statement

As IPv6 transition gradually goes on, native IPv6 networks will start to appear, including newly-built native IPv6 backbone networks, enterprise networks upgraded to native IPv6, and newly-built native IPv6 enterprise networks. Meanwhile, the upgrading process for IPv4 applications will take a lot of time. So, IPv4 communication requirements will appear among these networks. The hosts will be in a native IPv6 network, but have IPv4 applications on the upper layer. So, it would be helpful if we can make the IPv4 applications on these hosts communicate with each other.

Furthermore, if we can setup an architecture where IPv6 is used as a unified middle layer, it will have some advantages and accelerate the IPv6 transition. The new architecture will make network infrastructure transition transparent to the upper layer. And it also brings IPv4 applications into IPv6 network, which will help with the problem that there aren't quite enough IPv6 applications to use the IPv6 networks yet. So, from both the point of application and transition architecture, it is worthwhile to carry out our research on supporting IPv4 applications in native IPv6 networks.

Traditional tunnel mechanisms like IPv4 over IPv6 [1] [2] give a way to transfer IPv4 data over IPv6 networks. However, in order to set up a tunnel, each end has to know the IPv4 address of the other tunnel end. This address is usually manually configured, or gained from the DNS. But in the scenario described in Section 2, both ends are in native IPv6 network and won't have an IPv4 address at all. So, our mechanism solves such a problem: If both ends only have IPv6 addresses, how can their legacy IPv4 applications keep working? We do plan to use IPv4 over IPv6 tunnels as part of our mechanism for data transfer, but first we have to design a mechanism to help set up such tunnels. And this is the chief contribution of our work.

3. Related Work

Among current transition mechanisms, there are three related to IPv4 application: BIS, BIA and DSTM. We will give a brief description of each mechanism, and analyze them in the scenario described in Section 2.

0-7695-3006-0/07 $25.00 © 2007 IEEE
DOI 10.1109/IPC.2007.43

3.1 BIS, BIA and DSTM

BIS [3] and BIA [4] are two mechanisms for IPv4 applications. They both contain a name resolution and an address mapping module to handle all functions related to IPv4 address. During DNS resolution, an IPv4 address is randomly generated for applications in the upper layer. IPv4 data packets are converted into IPv6, and transferred through IPv6 infrastructure. The main difference between them is that BIA provides an API mapping function, which can automatically call corresponding IPv6 APIs when an IPv4 API is called.

DSTM [5] mechanism consists of two parts: a DSTM server with an IPv4 global address pool, and several DSTM TEPs (Tunnel End Point) on the IPv6 network border. When a host wants to communicate with an IPv4 host, it will contact DSTM server to apply for a temporary IPv4 global address. The server responds with an address and some information to set up a tunnel between the host and a DSTM TEP. IPv4 packets are encapsulated in IPv6 headers and transferred to TEP through the tunnel. The TEP de-encapsulates the IPv4 packets, and passes them on to IPv4 networks.

3.2 Analysis

By analyzing BIS/BIA and DSTM, we can see that they aren't quite suitable for the scenario in Section 2.

In BIS/BIA, packets are converted between IPv4 and IPv6 with protocol translation. In the scenario described in Section 2, both applications use IPv4, so a packet needs to be translated twice, which might lead to low efficiency and synchronization problems. Another problem is that BIS and BIA are originally designed for IPv4-to-IPv6 communication, so they translate all incoming IPv6 packets into IPv4 packets. Since the hosts are in native IPv6 networks, they should have some IPv6 applications running, and then the IPv6 applications will be unable to receive any IPv6 packets.

In DSTM, the host will ask DSTM server for a temporary address only if it sees the other end's IPv4 address first. However, in the scenario in Section 2, both ends are in native IPv6 networks. Neither of them will have IPv4/IPv4-mapping record in the DNS, so the mechanism of DSTM won't work.

4. Tunnels-Between-Hosts Mechanism

4.1 System Structure

In order to make IPv4 applications work without any change on the program, we decide to modify the DNS part in the protocol stack to provide a temporary IPv4 address. To generate this address, we insert two modules: Address Negotiation (AN) and Mapping Table (MT) into the protocol stack. AN handles the negotiation with other hosts, and MT keeps all IPv4-IPv6 address mappings. When a temporary IPv4 address is decided, we can easily find corresponding IPv6 address in the mapping table. In other words, an IPv4-in-IPv6 tunnel between two hosts is automatically setup during the address negotiation. In the data plane, we use IPv4-in-IPv6 encapsulation to prevent protocol translation. We create a "virtual interface" in the host's stack. By adding certain entries into the host's routing table, we can deliver the IPv4 packets to this interface. The IPv4-in-IPv6 packets are then encapsulated and sent through tunnels. Incoming IPv4-in-IPv6 packets are also delivered to this interface, de-encapsulated and passed on to the upper layer.

4.2 Working Process

Our mechanism can be divided into three chief steps: DNS resolution, Address Negotiation and Data Packet Transfer. The detailed process is described as follows:

1. IPv4 application on PC A calls DNS to resolve the IPv4 address for PC B, and modified DNS module sends requests for both IPv4 and IPv6 addresses.
2. If only IPv6 entry exists, negotiation starts and each other's temporary IPv4 addresses are negotiated.
3. IPv4-IPv6 address mapping is recorded in the MT, and the IPv4 address is returned for application.
4. IPv4 packets are delivered to the virtual interface. The interface looks up corresponding IPv6 address in MT, encapsulates packets in IPv6 headers and sends them to PC B through IPv4-in-IPv6 tunnel.

4.3 Address negotiation

4.3.1 Why negotiation? In simple cases where the application only uses address for data transfer and nothing else, we can simply let the sender generate any IPv4 address as a symbol for the receiver, use IPv6 for transfer, and no negotiation is needed. Unfortunately, this is not always the case. Some applications use IPv4 address as useful data, like in the PORT command in FTP. In this case, if two senders generate the same IPv4 addresses for themselves, the IPv4 application on the receiver's side will be confused. And a lot of security problems may also begin to appear. So, the two ends should have an "agreement" on each other's address. And this calls for a negotiation process.

4.3.2 Load Balance. When it comes to detailed design of the negotiation protocol; there are two typical ways to choose from. One is to include both the initiator and receiver in the address assigning process. The other is to let initiator alone assign addresses, receiver just needs to check whether the results are acceptable. The former considers both sides' mapping tables, thus can generate addresses more efficiently. But it will increase

system states. The latter will make the system state easier, but may result in an increase in negotiation steps and a lower efficiency in address usage. In order to choose a proper mechanism, we designed the following experiment to compare the results.

In the experiment, we studied one-way and two-way negotiation separately under the same network topology and activity rules. We simulated the situations of both C/S communication mode and P2P mode. The number of steps during each negotiation is recorded, and the results are shown in figure 1.

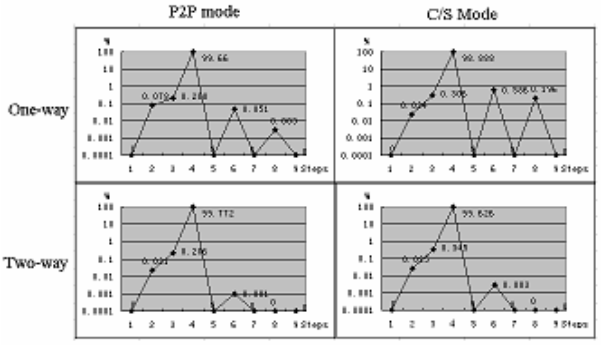

Fig. 1 Experiment Result 1

In the above figure, x-axis stands for the number of steps, and y-axis stands for its appearance frequency. There're around 600,000 times of negotiation during the experiment. From the result, we can see that two-way mechanism has advantage in average negotiation steps, because it has less frequency of steps above 6. This is especially obvious under C/S mode, which is due to the fact that server plays a complete passive role in the one-way mechanism. Clients generate addresses only according to their own mapping tables, which may result in a quick exhaustion of the server's available addresses and increase the probability of collision or failure. Considering this performance advantage, since two-way mechanism won't bring too much complexity, we decide to choose it for our negotiation process.

4.3.3 Finding the proper number. According to our experiment result, for a network of 2000 nodes, if we generate 3 addresses each time, the negotiation can be assured to finish in 4 steps. If we generate 2, there may be some need for 6 steps, but the percentage is quite small, less than 0.01% to be exact. If only one address is generated, this percentage grows by two orders of magnitude, but is still acceptable. Since two peers may both need addresses at the start of a communication, we decide to choose 2 as the number. In practice, this number can be adjusted according to the host's resource situation and network scales.

4.3.4 Protocol Description. After deciding on the two-way mechanism and 2 addresses each time, we can give a detailed description of our negotiation protocol. When a negotiation starts, we can divide the situation into four basic types according to the states of both peers.

1. Neither has any IPv4 address in use.
   S: send negotiation request
   R: respond with both sides' assigned addresses
   Negotiation finished.
2. Only receiver has IPv4 address in use.
   S: send negotiation request
   R: respond with R's address and assign one for S
   Negotiation finished.
3. Only initiator has IPv4 address in use.
   S: send negotiation request and S's address
   R: respond, asking S to assign one for R
   S: respond with the assigned address for R
   Negotiation finished.
4. Both have IPv4 addresses in use.
   S: send negotiation request, S's address and 2 generated addresses
   R: check if S's address or generated addresses causes a collision in the mapping table
   Respond with the result, R's address and 2 generated addresses
   S: check R's address and the generated addresses

After these two handshakes, S now knows enough information to make the following decisions:

If no collision is happening, both sides will use its current address
If collision happens on one side, use one of the generated addresses as that side's address
If 2 collisions happen, use two of the generated addresses as each side's address
If the number of usable generated addresses isn't enough, restart the negotiation

After S has made the decision, 2 more handshakes are needed between the two hosts for confirmation.

4.3.5 Scalability Analysis. Suppose the total number of available addresses is N. When A starts a connection with B, let its current address number be Aa, current connection number be Ca, then its mapping table size will be Aa+Ca. Likewise, the number for B will be Ab+Cb. In the worst situation where these two mapping tables have no common entry at all, the probability that at least one of A's current addresses is usable is $Pa = 1 - (\frac{Ab + Cb}{N})^{Aa}$. Likewise, the probability for B's address is $Pb = 1 - (\frac{Aa + Ca}{N})^{Ab}$. As for the 2 addresses A generated, each has a probability of $P = 1 - \frac{Ab + Cb}{N - Aa - Ca}$ to be usable, and $P' = 1 - \frac{Aa + Ca}{N - Ab - Cb}$ can be decided for those generated by B. With p and p', we can further

calculate the probabilities p_0~p_4, each corresponds to one situation that among the 4 generated addresses, 0/1/2/3/4 addresses are usable.

Because experiments showed that most negotiation can be finished in 4 steps, we only have to calculate the probability of NOT finishing in 4 steps, according to the protocol described above, this probability is.

$$P_{n>4} = 1 - P_a P_b - P_a(1-P_b)(1-P_0) - P_b(1-P_a)(1-P_0) - (1-P_a)(1-P_b)(1-P_0-P_1)$$

Deduction of this expression shows that its value changes in the same trend as two variables, (A+C)/N and (A+C)/(N-A-C). Since address number A is usually very small compared with the whole address number N and connection number C, we can claim that the value changes in the same trend as C/N. So, as long as the number of connections grows in the same scale as N, our mechanism will keep its performance.

In order to test the scale of network our mechanism can support, we carried out the following simulation. We fixed network topology as well as communication mode, and changed the rate of address number / host number. The rate is changed from 1:5 to 1:20, and the results are shown in figure 2.

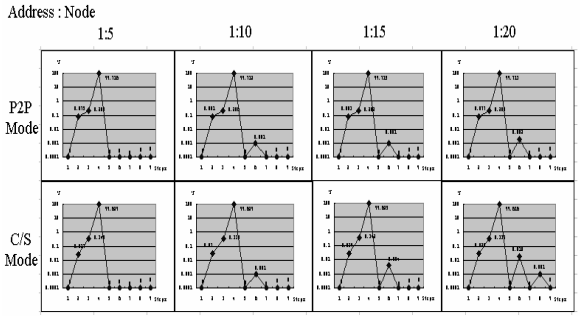

Fig.2 Experiment Result 2

We can see that the sub figures look quite alike, which means our mechanism doesn't lose its efficiency much as the network scale grows. In practice, we plan to use 169.254.xxx.xxx addresses as our address range. This means we have more than 65,000 addresses to use. With the scale of 1:20, our mechanism can be expected to support more than 1.3 million hosts. This proves that our mechanism has a pretty good scalability.

4.4 Comparison and Analysis

TBH provides an effective transition mechanism to resolve the IPv4-only application communication in native IPv6 network. The mechanism works directly between hosts, no special server is needed. Packets are directly transferred between two hosts without any relay, so problems like potential bottleneck can be avoided. IPv4 packets are encapsulated in IPv6 headers, so information loss during translation can be prevented. Finally, any two hosts using our mechanism can communicate with each other, so a fine deployment granularity can be achieved.

Table 1 gives a list of comparison among BIS, BIA, DSTM and TBH, showing that TBH is superior in most aspects.

Table 1. Comparison among three mechanisms

	BIS/BIA	DSTM	TBH
Architecture	No center	Server & TEP	No center
Data Transfer	Directly Between hosts	TEP Relay Potential Bottleneck	Directly Between hosts
IPv4/v6 Relation	Translation Possible Inform-loss	IPv4-in-IPv6 Tunnel	IPv4-in-IPv6 Tunnel
Connect Delay	No handshake	Authenticate & tunnel setup	Negotiation Around 4 Handshakes
Compatibility	Conflict with IPv6 application	Good	Good
Deployment	Fine granularity	Coarse granularity	Fine granularity

5. Conclusion

This paper presents a new mechanism to resolve the problem of supporting IPv4-only application in native IPv6 network. The comparison shows that the proposed method is the most effective. Currently, a prototype is implemented and we are conducting experiments on CERNET2, which is a nationwide native IPv6 network in China. The results show that the IPv4 applications can well communicate with each other in native IPv6 environment.

References

[1] M.Blanchet, "DSTM IPv4 over IPv6 tunnel profile for Tunnel Setup Protocol(TSP)", draft-blanchet-ngtrans-tsp-dstm-profile-00, February 2002.

[2] Wu Jian-ping,Li Xing,Cui Yong,et al. "4over6:IPv4 natwork interconnection over IPv6 backbone without explicit tunneling" [J].Acta Electronica Sinica,2006(3).

[3] K.Tsuchiya, H.Higuchi, Y.Atarashi, "Dual Stack Hosts using the 'Bump-In-the-Stack' Technique (BIS)", RFC2767, February 2000.

[4] S.Lee, M-K.Shin, Y-J.Kim, E.Nordmark, A.Durand, "Dual Stack Hosts Using 'Bump-in-the-API' (BIA)", RFC3338, October 2002.

[5] J. Bound et al., "Dual Stack Transition Mechanism (DSTM)", draft-ietf-ngtrans-dstm-08, Mar, 2002.