Abstract

This document describes an IPv6/IPv4 gateway mechanism that is based on the SOCKS protocol [SOCKSv5]. By enhancing the SOCKS mechanism to support the heterogeneous communications relays between IPv6 and IPv4 at the SOCKS server, the SOCKS-based IPv6/IPv4 gateway mechanism is accomplished.

The SOCKS-based gateway mechanism enables the IPv6 nodes and IPv4 nodes to communicate with each other smoothly without sacrificing any conveniences and functionalities of current communication methods.
1. Introduction

This document describes an IPv6/IPv4 gateway mechanism that is based on the SOCKS protocol [SOCKSv5]. By enhancing the SOCKS mechanism to support heterogeneous communications, the SOCKS-based IPv6/IPv4 gateway mechanism is accomplished.

The SOCKS-based gateway mechanism is designed to satisfy the following requirements.

1. Maintain the conveniences of current IPv4 communication methods and frameworks of existing communication networks

2. Utilize existing user applications that are designed only for IPv4 communications to heterogeneous communications without modifying them.

2. Be scalable and able to support the translations for all typical communication services.

Since the SOCKS-based IPv6/IPv4 gateway mechanism is based on relaying two "terminated" connections, most problems of the NAT-based translator mechanisms are eliminated.

Another advantage of the mechanism is that there is no need to modify the DNS system, because it has a mechanism that enable the DNS name resolving actions of SOCKS clients to be delegated to an enhanced SOCKS server that has both IPv6 and IPv4 stacks.

This DNS name resolving delegation mechanism is realized by modifying (socksifying) client applications. This is a constraint. To eliminate the need to modify the DNS system, it is necessary to socksify the applications. Thus, there is a dilemma in that either socksification or DNS modification is necessary.

Also, the mechanism has the following additional by-product benefits. Since the SOCKS-based gateway mechanism has much flexibility, it can support various topologies that mix IPv4 and IPv6 communications. In case of multiple chained relay topologies, the SOCKS-based gateway mechanism can realize similar topologies that are supported by packet tunneling techniques. Compared with other translator mechanisms, the SOCKS-based gateway mechanism has the following benefits. There is no packet fragmentation vulnerability, and the connection can be authenticated by the native SOCKS authentication methods.

2. Basic SOCKS-based Gateway Mechanism

The Fig. 1 shows the basic SOCKS-based gateway mechanism.

H. Kitamura et al.
In this figure, the Client C initiates the communication to the Destination D. Two new functional blocks that compose the system are introduced.

One, *Socks Lib*, is introduced into the client side (Client C) (this procedure is called "socksifying"). It is located between application layer and socket layer, and can replace applications' socket APIs and DNS name resolving APIs (e.g., gethostbyname(), getaddrinfo() etc.) dynamically. A mapping management table exists in it for "DNS name resolving delegation" mechanism (described below). Each socksified application has its own *Socks Lib*.

The other, *Gateway*, is an enhanced SOCKS server. It is installed on the IPv6 and IPv4 dual stack node (Gateway G). All types of protocol combinations are relayed there between Client C (IPvX) and Destination D (IPvY). The *Socks Lib* invokes it. Each *Gateway* takes charge of one socksified connection.
The following four types of combinations of IPvX and IPvY are possible in this mechanism.

<table>
<thead>
<tr>
<th>Type</th>
<th>IPvX</th>
<th>IPvY</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>IPv4</td>
<td>IPv4</td>
<td>homogeneous (normal SOCKS)</td>
</tr>
<tr>
<td>B</td>
<td>IPv4</td>
<td>IPv6</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>C</td>
<td>IPv6</td>
<td>IPv4</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>D</td>
<td>IPv6</td>
<td>IPv6</td>
<td>homogeneous</td>
</tr>
</tbody>
</table>

Type A is supported by the normal SOCKS mechanism. Type B and C are the main targets for the SOCKS-based gateway mechanism. They provide heterogeneous communication topologies. Type D can be supported by the natural extension of the SOCKS mechanism, because it is a homogeneous communication.

The *Socks Lib* communicates with the *Gateway* by using SOCKS protocol [SOCKSv5]. The connection between the Client C and the Gateway G is called a "socksified connection." It can transfer not only data but also control information (e.g., location information of the Destination, etc.).

In order to provide appropriate destination information to the application on the Client C, the *Socks Lib* also replaces other types of socket APIs (e.g., getpeername(), etc.) that provide information about the destination, and necessary information for this replacement is transferred via the socksified connection. This mechanism provides the same usability of current communication methods to the applications on the Client C, and it is not necessary to modify the applications at all to utilize this mechanism.

The connection between the Gateway G and the Destination D is an ordinary connection. Server applications on Destination D understand that the source of the connection is the Gateway G (not Client C). It is not necessary to modify (socksify) them.

3. DNS Name Resolving Procedure

As [TRANSMECH] mentioned, it is essential for the transition mechanisms to cooperate with the DNS name resolving mechanism. The SOCKS-based gateway mechanism has the capability to cooperate with it without modifying the DNS system. This function is one of the big advantages of the SOCKS-based gateway mechanism.

In this section, the procedure of how the SOCKS-based gateway mechanism cooperate with the DNS name resolving mechanism is described.
1. An application on the source terminal (Client C) tries to get the IP address information of the destination terminal (Destination D) by calling the DNS name resolving function (e.g., gethostbyname2()). At this time, the logical host name ("FQDN") information of the Destination D is passed to the *Socks Lib* as an argument of called APIs.

2. Since the *Socks Lib* has replaced such DNS name resolving APIs, the real DNS name resolving APIs is not called here. The argued "FQDN" information is merely registered into a mapping management table of *Socks Lib*, and a "fake IP" address is selected as a reply value from a reserved special IP address space that is never used in real communications. The "fake IP" address must be suitable for requests called by the applications. Namely, it must belong to the same address family of the Client C, even if the address family of the Destination D is different from it. After the selected "fake IP" address is registered into the mapping management table as a pair with the "FQDN", it is replied to the application.

3. The application receives the "fake IP" address, and prepares a socket. The "fake IP" address information is used as an element of the "socket". The application calls socket APIs (e.g., connect()) to start a communication. The "socket" is used as an argument of them.

4. Since the *Socks Lib* has replaced such socket APIs, the real socket function is not called. The IP address information of the argued socket is checked. If the address belongs to the special address space for the fake address, the matched registered "FQDN" information of the "fake IP" address is obtained from the mapping management table.

5. The "FQDN" information is transferred to the *Gateway* on the relay server (Gateway G) by using the SOCKS command that is matched to the called socket APIs. (In case of connect(), the CONNECT command is used.)

6. Finally, the *Gateway* calls the real DNS name resolving APIs (e.g., gethostbyname2()). At this time, the received "FQDN" information is used as an argument of the called APIs.

7. The *Gateway* receives the "real IP" address from a DNS server, and creates a "socket", The "real IP" address information is used as an element of the "socket".

8. The *Gateway* calls socket APIs (e.g., connect()) to communicate with the Destination D. The "socket" is used as an argument of them.

By using this mechanism, DNS name resolving actions are delegated.
from the source terminal (Client C) to the relay server (Gateway G). Thus the mechanism is called "DNS name resolving delegation."

* DNS Name Resolving Delegation and address mapping

The advantages of the "DNS name resolving delegation" mechanism come from the fact that the DNS name resolving actions are taken at the relay server (Gateway G). Since the relay server is an IPv4 and IPv6 dual stack node, DNS name resolving queries for any types of destinations can be done without causing problems. It is not necessary to modify the existing DNS mechanism at all.

Without this mechanism, an IPv4 application can not resolve the host name of an IPv6 destination. Even if it can get the IPv6 address by some method, it is impossible for the IPv4 application to deal with such an IPv6 address. It is impossible because the IPv6 address is four times longer than the IPv4 address, and the IPv4 application does not have any long address space in which to store it.

Using this mechanism brings forth an additional advantage: an IPv4 application can also use a numerical IPv6 address expression to specify an IPv6 destination terminal. Since the IPv6 address expression includes colons ("::"), it is identified as an FQDN expression (not a numerical IP expression) for the IPv4 application. Thus the numerical IPv6 address expression is treated the same as the DNS expression in the "DNS name resolving delegation" mechanism.

The problem with the mechanism is that a failure of the DNS name resolving process is detected incorrectly at the source terminal (Client C). It is detected as a failure of the connection creation. In order to solve this problem, the author has proposed extensions of the current SOCKS protocol [SOCKS5EXT].

* Address Translation and Mapping

One of the good characteristics of the SOCKS-based gateway mechanism is its ability to manage address translation and mapping wisely.

In case of the NAT-based transparent model translator mechanism (e.g., [NATPT]), it is necessary to reserve global and wide address space for the address mapping. Also, complex address allocation and management mechanisms (by introducing address mappers, etc.) are needed.

In case of the SOCKS-based gateway mechanism, such complex mechanisms are not necessary, because corresponding address mapping is done at the *Socks Lib* by using the fake IP address and the mapping
management table. The mapping management table is prepared on each
application. It is locally closed and independent from others.
Therefore, it is easy to manage the table, and it is not necessary to
reserve global and wide address space.

As [CATTRANS] mentioned, the SOCKS-based gateway mechanism is free of
the address mapping problem.

4. Multiple Chained Relay Mechanism (Advanced usage)

The SOCKS-based gateway mechanism has the flexibility to support
multiple chained relay topologies. With the mechanism, IPv4 and IPv6
mixed various communication topologies are realized.

The Fig. 2 shows the structure of the multiple chained relay
mechanism.

---

Client C       Gateway G1       Gateway G2    Destination D
+-----------+     (Server 1)       (Server 2)
|Application|                 |Application|
+-----------+     +------------+     +-----------+
| *SOCKS Lib* | | *Gateway1* | | *Gateway2* | | Application|
| Socket DNS | | Socket DNS | | Socket DNS | | Socket DNS|
+-------------+ +---------------+ +---------------+ +-----------+
| [ IPv X ] | | [IPvX] (IPvY)| | (IPvY) (IPvZ) | | { IPv Z } |
+-----------+ +---------------+ +---------------+ +-----------+
|Network I/F| | Network I/F| | Network I/F| | Network I/F|
+-----------+ +---------------+ +---------------+ +-----------+
|          | |          | |          | |          |
|           | |          | |          | |          |
|socksified| |socksified| |normal | |socksified|
|connection| |connection| |connection| |connection|
| (ctrl)+data| | (ctrl)+data| |data only| | (ctrl)+data|

Fig. 2 Multiple Chained Relay Mechanism

In this figure, the source terminal (Client C) initiates the
communication with the destination (Destination D). Underneath, the
connection is replaced with three connections, and they are relayed
at the two relay servers (Gateway G1 and G2). The *Gateway* includes
the same type of the functions of *Socks Lib*. By enabling the *Socks
Lib* functions at the *Gateway1* on the first relay server (Gateway
G1), the multiple chained relay topology is realized.

There is no limitation on the number of relay operations between the

H. Kitamura et al. [Page 7]
source terminal and the destination terminal. It is possible to have more than two intermediate relay servers. To simplify the explanation, twice-relayed topology is shown here. All types of the protocol relay combinations are possible.

In case of twice-relayed topology, the following eight types of combinations of IPvX, IPvY, and IPvZ are possible.

<table>
<thead>
<tr>
<th>Type</th>
<th>IPvX</th>
<th>IPvY</th>
<th>IPvZ</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>IPv4</td>
<td>IPv4</td>
<td>IPv4</td>
<td>homogeneous (normal SOCKS)</td>
</tr>
<tr>
<td>F</td>
<td>IPv4</td>
<td>IPv4</td>
<td>IPv6</td>
<td>heterogeneous *</td>
</tr>
<tr>
<td>G</td>
<td>IPv4</td>
<td>IPv6</td>
<td>IPv4</td>
<td>heterogeneous * (tunnel-like)</td>
</tr>
<tr>
<td>H</td>
<td>IPv6</td>
<td>IPv6</td>
<td>IPv4</td>
<td>heterogeneous *</td>
</tr>
<tr>
<td>I</td>
<td>IPv6</td>
<td>IPv6</td>
<td>IPv4</td>
<td>heterogeneous *</td>
</tr>
<tr>
<td>J</td>
<td>IPv6</td>
<td>IPv6</td>
<td>IPv4</td>
<td>heterogeneous * (tunnel-like)</td>
</tr>
<tr>
<td>K</td>
<td>IPv6</td>
<td>IPv6</td>
<td>IPv6</td>
<td>heterogeneous *</td>
</tr>
<tr>
<td>L</td>
<td>IPv6</td>
<td>IPv6</td>
<td>IPv6</td>
<td>homogeneous</td>
</tr>
</tbody>
</table>

Type E is supported by the normal SOCKS mechanism. The types from F to K are the main targets for the gateway mechanisms. They provide heterogeneous communication topologies. Type L can be supported by the natural extension of the SOCKS mechanism, because it is a homogeneous communication.

Type G and type J can provide interesting topology cases. They are similar to the topologies that the tunneling techniques can provide. Compared with the tunneling techniques, the SOCKS-based gateway is advantageous, because it is composed of relays of real socket connections. This mechanism does not have to suffer from the following problems that are caused by tunneling techniques.

* Fragmentation vulnerability

The tunneling technique uses the packet encapsulation mechanism. In the encapsulation mechanism, the packet length is changed because of the additional header, and it may exceed the MTU of the network. Thus, the tunneling technique is vulnerable in that the packet might be fragmented. With the SOCKS-based gateway mechanism, there is no fragmentation vulnerability because the packet size is adjusted at the application layer.

As an added advantage, well-authenticated connections are provided. This is because the SOCKS mechanism is originally designed for firewall systems, and it has various authentication methods.

H. Kitamura et al.
5. Characteristics of the SOCKS-based gateway mechanism

In this section, the characteristics of the SOCKS-based gateway mechanism are summarized.

1. DNS modification is not necessary:

   It is not necessary to reserve global and wide address space for address mapping.

   Also, address map servers are not necessary, because there is a "DNS Name Resolving Delegation" mechanism.

2. Application independent:

   If the applications use socket APIs and DNS name resolving APIs, the gateway mechanism is applied to them, since the mechanism is realized by replacing these APIs that at the *Socks Lib*.

   As most communication programs use these APIs, there is no application dependency.

   (Applications that exchange IP address information with peers are exceptions)

3. OS and NIC types independent:

   Since the *Socks Lib* and the *Gateway* run at the application layer, the SOCKS-base gateway mechanism runs on any platforms of either UNIX or Windows OSs, and there is no dependency on the types of physical NICs.

4. Only an easy socksification procedure is necessary:

   It is easy to install the *Socks Lib* (socksify) to the source terminal, because the dynamic link library technique helps this procedure.

5. IPv6 new features (e.g., IPSEC) can be utilized easily:

   Since connections are first terminated and then relayed at the *Gateway*, it is easy to enable IPv6 new features on the IPv6 side.

   With NAT-based translators, it is impossible to do the same thing.

6. Current existing client SOCKSv5 library can be used:
In case of the IPv4 -> IPv6 relay, the current existing client
SOCKSv5 library that is designed for IPv4 -> IPv4 communication
can be used without modification.

7. Both TCP and UDP relays are possible:

Since the SOCKS protocol supports both TCP and UDP relays, this
mechanism can also relay both TCP and UDP connections.

8. Both IPv4 -> IPv6 and IPv6 -> IPv4 relays are possible:

This mechanism can realize not only the IPv4 -> IPv6 relay but
also the IPv6 -> IPv4 relay with the same method.

9. Multiple chained relays are possible:

Since the mechanism has flexibility, all types of protocol
combinations of multiple chained relays are possible.

10. Can support exceptional applications that exchange IP address
    information at the application level:

    It is easy for the SOCKS-based gateway mechanism to introduce
    special management routines. If such protocols as ftp are known,
    the SOCKS-based gateway mechanism can support them by introducing
    special protocol translation routines.

    (Implementations of the SOCKS-based gateway mechanism have
    supported the ftp protocol translation [see Appendix A])

11. Easy load balancing:

    Since the *Gateway* is installed as a server, it is easy to
    balance the load by supplying multiple servers.

6. Constraints

The SOCKS-based gateway mechanism requests socksification of
applications (install *Socks Lib*) to realize heterogeneous
communications. Though the socksification is not difficult, it is
one of the constraints for users.

The socksification mechanism provides the advantage that the SOCKS-
based gateway mechanism does not require modification of the DNS
system. Thus, there is a dilemma in that either socksification or DNS
modification is necessary.
Other than the socksification, the SOCKS-based gateway mechanism has the following three types of constraints.

1. Essential constraints:

   Constraints are caused by the address length difference between IPv4 and IPv6.

   Functions that request an IP address as one of the return values (e.g., getpeername() and getsockname() etc.) can not provide the correct IP address as a return value. However, an appropriate port value can be provided, because IPv4 and IPv6 use the same size port space and necessary port information is transferred by the SOCKS protocol.

   From a realistic viewpoint, this is a minor constraint, because such functions are called in order to get port information (not IP address information).

2. Limitation of the SOCKS mechanism:

   Since the current SOCKS system can not socksify all of the tricky applications in which extraordinary means are used to create connections, the SOCKS-based gateway mechanism can not be applied to them.

3. Constraints to deal with the fake address:

   The fake address must be dealt with as a temporary value at the application; it must not be recorded permanently as a bookmark, etc. After the application is finished, the fake address information must be released. Otherwise, problems will happen.

   From a realistic viewpoint, this is also a minor constraint, because most applications record FQDN information instead of fake addresses.

   It is theoretically possible for the SOCKS mechanism to support ICMP relays, but the current SOCKS mechanism does not. So, the SOCKS-based gateway mechanism does not support ICMP relays.

   However, the reference implementation of the current SOCKS mechanism support typical ICMP applications (ping and traceroute) by using different methods. So, the SOCKS-based gateway mechanism can support ping and traceroute.

   (Implementations of the SOCKS-based gateway mechanism have supported ping and traceroute [see Appendix A])
7. Experiments

We have been testing SOCKS64, one of the SOCKS-based gateway implementations in many environments. Since July 1998, we have been running the SOCKS64 server in between the 6bone and the internal network of Fujitsu Ltd., Japan. Computers in Fujitsu’s internal network already have been "socksified", so users can connect to 6bone hosts merely selecting our SOCKS64 server as their SOCKS gateway.

For an interoperability test, we have been providing a SOCKS64 server in the WIDE Project [WIDEV6WG] Camp, where about 200 Internet researchers are participating in the camp. A temporary network called "camp-net" is constructed at the camp for a variety of experiments and provides services to the attendees. More than a hundred hosts are connected to the camp-net.

In these environments, we have tested many types of SOCKS5 clients, including the SocksCap32 [SOCKSCAP] on Windows and a variety of UNIX based SOCKS libraries, and confirmed any clients successfully connected to the IPv6 hosts using telnet, ftp, http, ssh.

Also we have tested very complicated interconnection of IPv4 and IPv6. At the WIDE camp held in September 1998, we constructed an IPv4 over IPv6 tunnel using the SOCKS64 and the FAITH translator. In this system, the SOCKS5 clients had no problems connecting to the IPv4 hosts through the IPv6 networks.

IPv4 host --SOCKS-- SOCKS64 --IPv6-- IPv6-IPv4 -- IPv4 host
/IPv4 server translator

In conclusion, SOCKS64 provides an easy and sure way to let IPv4 hosts connect to IPv6 hosts.

H. Kitamura et al.
8. Conclusion

As described above, the SOCKS-based IPv6/IPv4 gateway mechanism has many advantages. From a realistic viewpoint, constraints are negligible.

Only two things are necessary to utilize this gateway mechanism.

1. Prepare an enhanced SOCKS sever on the dual stack node. [see Appendix A. two implementations are available]
2. Socksify applications when you execute them.
   This is a very easy procedure.

You don’t have to modify the DNS system at all.
You don’t have to reserve any addresses.
You don’t have to prepare address mapping servers, etc.

Since the mechanism is based on relaying two "terminated" connections, most problems of the NAT-based translator mechanisms are eliminated.

Also, the "DNS name resolving delegation" mechanism works effectively. It enables smooth heterogenous communications between IPv6 and IPv4 nodes without modifying the DNS system.

9. Changes

This draft is based on <draft-jinzaki-socks64-00.txt> [SOCKS64] and <draft-kitamura-socks-ipv6-trans-arch-00.txt> [SOCKSTRANS].

Appendix A. Implementations

Currently, there are two independent implementations of the SOCKS-based IPv6/IPv4 gateway mechanism. Both of them are open to the public.

One is Fujitsu Lab.’s implementation, which is called "SOCKS64." Its source codes are available at the following URL.
ftp://ftp.kame.net/pub/kame/misc/socks64-v10r8-19990118.tgz

The other is NEC’s implementation. Its source codes are available at the following URL.
http://www.socks.nec.com/
INTERNET-DRAFT       A SOCKS-based IPv6/IPv4 Gateway          April 1999

References

[SOCKS64] A. Jinzaki, S. Kobayashi, "SOCKS64: An IPv4-IPv6 intercommunication gateway using SOCKS5 protocol," <draft-jinzaki-socks64-00.txt> work in progress


[NATPT] G. Tsirtsis and P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)", <draft-ietf-ngtrans-natpt-05.txt> work in progress


Authors’ Addresses:

Hiroshi Kitamura
NEC Corporation
C&C Media Research Laboratories
1-1, Miyazaki, 4-Chome, Miyamae-ku,
Kawasaki, Kanagawa, 216-8555, JAPAN

Phone: +81 (44) 856-2123
Fax: +81 (44) 856-2230
Email: kitamura@ccm.cl.nec.co.jp