Network Working Group
Request for Comments: 3089
Category: Informational

H. Kitamura
NEC Corporation

April 2001

A SOCKS-based IPv6/IPv4 Gateway Mechanism

Status of this Memo

Copyright Notice

Abstract

This document describes a SOCKS-based IPv6/IPv4 gateway mechanism that enables smooth heterogeneous communications between the IPv6 nodes and IPv4 nodes.

It is based on the SOCKS protocol [SOCKSv5]. By applying the SOCKS mechanism to the heterogeneous communications and relaying two "terminated" IPv4 and IPv6 connections at the "application layer" (the SOCKS server), the SOCKS-based IPv6/IPv4 gateway mechanism is accomplished.

Since it is accomplished without introducing new protocols, it provides the same communication environment that is provided by the SOCKS mechanism. The same appearance is provided to the heterogeneous communications. No conveniences or functionalities of current communications are sacrificed.

1. Introduction

The SOCKS-based IPv6/IPv4 gateway mechanism is based on a mechanism that relays two "terminated" IPv4 and IPv6 connections at the "application layer" (the SOCKS server); its characteristics are inherited from those of the connection relay mechanism at the application layer and those of the native SOCKS mechanism.

2. Basic SOCKS-based Gateway Mechanism

Figure 1 shows the basic SOCKS-based gateway mechanism.

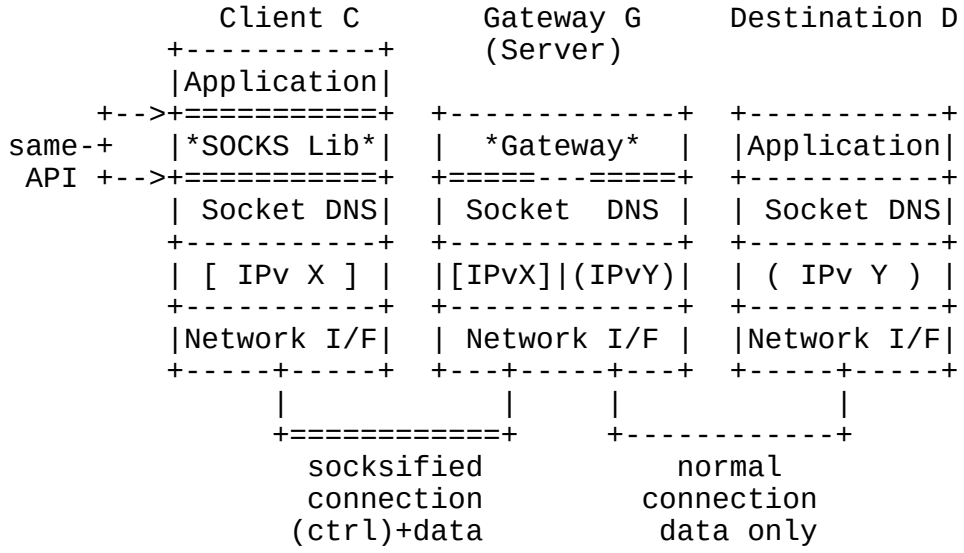

Fig. 1 Basic SOCKS-based Gateway Mechanism

In this figure, the Client C initiates the communication to the Destination D. Two new functional blocks are introduced and they compose the mechanism.

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

The other, *Gateway*, is installed on the IPv6 and IPv4 dual stack node (Gateway G). It is an enhanced SOCKS server that enables any types of protocol combination relays between Client C (IPvX) and Destination D (IPvY). When the *Socks Lib* invokes a relay, one corresponding *Gateway* process (thread) is spawned from the parent *Gateway* to take charge of the relay connection.

The following four types of combinations of IPvX and IPvY are possible in the mechanism.

```
 type C ------ G ------ D
        [IPvX]   (IPvY)
  A      IPv4     IPv4       homogeneous (normal SOCKS)
  B      IPv4     IPv6     * heterogeneous *
  C      IPv6     IPv4     * heterogeneous *
  D      IPv6     IPv6       homogeneous
```

Type A is supported by the normal SOCKS mechanism. Type B and C are the main targets for the SOCKS-based IPv6/IPv4 gateway mechanism. They provide heterogeneous communications. Type D can be supported by the natural extension of the SOCKS mechanism, because it is a homogeneous communication.

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

The connection between the Gateway G and the Destination D is a normal connection. It is not modified (socksified). A server application that runs on Destination D does not notice the existence of the Client C. It recognizes that the peer node of the connection is the Gateway G (not Client C).

No new protocols are introduced to the SOCKS protocol [SOCKSv5] to accomplish the mechanism.

* Packet Size Adjustment

  Since the length of the IPv6 header is different from that of the IPv4 header, it is necessary to consider the packet size adjustment in heterogeneous communications. If this is not taken into consideration, the packet size may exceed the MTU of the network.

  In the SOCKS-based IPv6/IPv4 gateway mechanism, it never exceeds the MTU, because the mechanism is based on relaying two "terminated" connections at the "application layer". The relayed data is a simple data stream for the application, and the packet size is naturally adjusted at each relayed connection side.

* Authenticated Relay

  Since the SOCKS is originally designed for firewall systems and it has various authentication methods, the relayed connections can be authenticated by the native SOCKS authentication methods.

3. DNS Name Resolving Procedure

In all communication applications, it is a necessary to obtain destination IP address information to start a communication. It is, however, theoretically impossible for the heterogeneous communications to obtain correct information, because an existing IPv4 application can not deal with an IPv6 address. It prepares only a 4-byte address space to store an IP address information, and it can not store an IPv6 address information into there. This is a critical problem caused by differences in address length.

In order to solve the problem, a feature called "DNS name resolving delegation" is used in the SOCKS-based IPv6/IPv4 gateway mechanism. The feature involves the delegating of DNS name resolving actions at the source node (Client C) to the relay server (Gateway G). Since the relay server is an IPv4 and IPv6 dual stack node, DNS name resolving queries for any address family types of destinations can be made without causing any problems. Therefore, it is not necessary to modify the existing DNS mechanism at all.

The feature supports not only the case in which a destination logical host name (FQDN) information is given but also the case in which a destination literal (numerical) IP address is given. The latter case is supported in almost the same way as the former case. Since the literal IPv6 address expression includes colons (":"), it is identified as an FQDN (not a literal IPv4 address) for the IPv4 application.

The SOCKS protocol specification [SOCKSv5] defines that IPv4 address, IPv6 address, and DOMAINNAME (FQDN) information can be used in the ATYP (address type) field of the SOCKS protocol format. In the "DNS name resolving delegation" feature, the DOMAINNAME (FQDN) information is used in the ATYP (address type) field. The FQDN information is transferred from the Client C to the Gateway G to indicate the Destination D.

In order to solve the formerly explained critical problem, an appropriate "fake IP" address is introduced in the feature, and it is used as a virtual destination IP address for a socksified application. A mapping table is also introduced in the *Socks Lib* (at the Client C) to manage mappings between "fake IP" and "FQDN". A "fake IP" address is used as a key to look up the corresponding "FQDN" information. The mapping table is local and independent of other applications or their *Socks Lib*s.

The transparentness to applications is maintained in the feature. Nothing special is required to execute it except socksifying the applications. Since DNS name resolving APIs are replaced by the *Socks Lib*, the "DNS name resolving delegation" is executed internally merely by calling the DNS name resolving APIs in ordinal methods.

The "DNS name resolving delegation" is accomplished only when FQDN information is used in the ATYP (address type) field of the SOCKS command. Therefore, it is mandatory to do so for heterogeneous communications. The method of using FQDN information in the ATYP field depends on the configuration setting and implementation of the SOCKS protocol. In order to simplify the discussion, only the case in which the FQDN information is used in the ATYP field is discussed here.

The detailed internal procedure of the "DNS name resolving delegation" and address mapping management related issues are described as follows.

1. An application on the source node (Client C) tries to get the IP address information of the destination node (Destination D) by calling the DNS name resolving function (e.g., gethostbyname()). At this time, the logical host name ("FQDN") information of the Destination D is passed to the application's *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 table in *Socks Lib*, and a "fake IP" address is selected as information that is replied to the application from a reserved special IP address space that is never used in real communications (e.g., 0.0.0.x). The address family type of 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 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 the APIs.

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 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. (e.g., for connect(), the CONNECT command is used.)

6. Finally, the real DNS name resolving API (e.g., getaddrinfo()) is called at the *Gateway*. At this time, the received "FQDN" information via the SOCKS protocol is used as an argument of the called APIs.

7. The *Gateway* obtains 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 the APIs.

The problem with the feature is that failures of the DNS name resolving process are detected incorrectly at the source node (Client C). They are detected as connection-establishment failures.

(Restrictions on applicability of "fake IP" address, etc., are described in Section 5.)

* Operations for Address Management (reservation, mapping etc.)

The SOCKS-based gateway mechanism does not require the reserving of a wide global address space for the address mapping, and complex address allocation and garbage-collection mechanisms are not necessary.

Such address management operations are done at the *Socks Lib* by using the fake IP address and the mapping table for the DNS name resolving delegation. Since the mapping table is prepared in each application, it is locally closed and independent of other applications. Therefore, it is easy to manage the table, and it is not necessary to reserve a wide global address space.

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 accomplished.

Figure 2 shows the structure of the multiple chained relay mechanism.

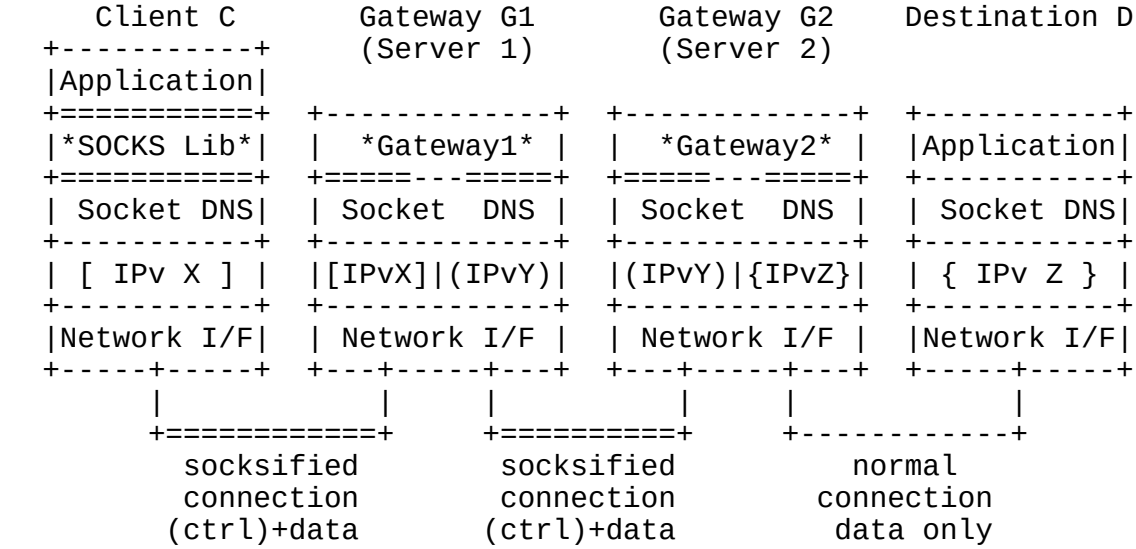

Fig. 2 Multiple Chained Relay Mechanism

In this figure, the source node (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 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 accomplished.

There is no limitation on the number of relay operations between the source node and the final destination node. It is possible to have more than two intermediate relay servers. To simplify the explanation, a twice-relayed topology is shown here.

Since the multiple chained relay is more complex than one-time relay and causes complexity, it is recommended that the multiple chained relay communication should be used only when it is necessary for some reason (e.g., usable protocols or topologies are limited by routers etc.).

5. Applicability statement

The SOCKS-based gateway mechanism requests socksification of applications (install *Socks Lib*) to accomplish heterogeneous communications. It is not necessary to modify (change source codes and recompile them, etc.) the applications, because typical socksification is done by changing the linking order of dynamic link libraries (specifically, by linking the SOCKS dynamic link library before the dynamic link libraries for normal socket and DNS name resolving APIs).

The mechanism does not request modification of the DNS system, because the DNS name resolving procedure at the Client C is delegated to the dual stack node Gateway G.

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, a suitable port value can be provided, because IPv4 and IPv6 use the same size port space and an appropriate port information is transferred by the SOCKS protocol.

2. Constraints of the SOCKS mechanism:

   Since the current SOCKS system can not socksify all of the tricky applications in which extraordinary manners 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 is used as a key value in the mapping table for the "DNS name resolving delegation" feature. When the application is finished and the mapping table disappears, the fake address information must be also released.

   Even if it is recorded permanently (e.g., recorded as a bookmark), serious problems will not occur. The recorded fake address information will merely become useless, because fake address

information is taken from a reserved special IP address space that is never used in real communications (e.g., 0.0.0.x) and such a information is useless for the normal communication applications. Furthermore, such cases will be rare because most applications usually record FQDN information (not fake IP address information) to the bookmark, etc.

5.1 Native SOCKS mechanism considerations

The characteristics of the SOCKS-based IPv6/IPv4 gateway mechanism are inherited from those of the native SOCKS mechanism. Therefore, consideration issues of the native SOCKS mechanism are discussed in this section.

The SOCKSv5 protocol is composed of three commands (CONNECT, BIND and UDP ASSOCIATE). All of three commands can be applied in the SOCKS-based IPv6/IPv4 gateway mechanism.

This document is described with assuming the usage of the CONNECT command mainly, because the CONNECT command is the main and most frequently used command in the SOCKS mechanism. Since the CONNECT command does not have clear week points, we can use it freely without considerations.

The other (BIND and UDP ASSOCIATE) commands have the following weak points. So, we have to consider these points when we use the BIND or UDP ASSOCIATE commands in the mechanism.

The BIND command is basically designed to support reverse-channel rendezvous of the FTP type applications. So, general usages of the BIND command may cause problems.

The UDP ASSOCIATE command is basically designed for simple UDP applications (e.g., archie). It is not general enough to support a large class of applications that use both TCP and UDP.

6. Security Considerations

Since the SOCKS-based IPv6/IPv4 gateway mechanism is based on SOCKSv5 protocol, the security feature of the mechanism matches that of SOCKSv5. It is described in the Security Considerations section of the SOCKS Protocol Version 5 [SOCKSv5].

The mechanism is based on relaying two "terminated" connections at the "application layer". The end-to-end security is maintained at each of the relayed connections (i.e., between Client C and Gateway G, and between Gateway G and Destination D). The mechanism does not provide total end-to-end security relay between the original source (Client C) and the final destination (Destination D).

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 NEC's implementation. Its source codes are available at the following URL.

http://www.socks.nec.com/

The other 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-...

References

[SOCKSv5] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D. and L. Jones, "SOCKS Protocol V5", RFC 1928, April 1996.

[TRANSMECH] Gilligan, R. and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 2893, August 2000.

[IPv6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[INET99] H. Kitamura, "Entering the IPv6 communication world by the SOCKS-based IPv6/IPv4 Translator", in Proceedings of INET99, July 1999.

Author's Address

Hiroshi Kitamura
NEC Corporation
Development Laboratories
(Igarashi Building 4F) 11-5, Shibaura 2-Chome,
Minato-Ku, Tokyo 108-8557, JAPAN

Phone: +81 (3) 5476-1071
Fax: +81 (3) 5476-1005
EMail: kitamura@da.jp.nec.com

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.