IPv6 Specific Issues to Track States of Network Flows

Yasuyuki Kozakai
Corporate Research & Development Center, Toshiba Corporation
yasuyuki.kozakai@toshiba.co.jp

Hideaki Yoshifuji
Graduate School of Media and Governance, Keio University
hideaki@yoshfuji.org

Hiroshi Esaki
Graduate School of Information Science and Technology, The University of Tokyo
hiroshi@wide.ad.jp

Jun Murai
Faculty of Environmental Information, Keio University
junsec@sfc.wide.ad.jp

ABSTRACT

Connection tracking subsystem on Linux tracks states of network flows. It is utilized by packet filter for stateful filtering.

In this paper, we propose solutions to issues that arise where connection tracking subsystem handles Routing Header and Mobile IPv6. We also describe how the current connection tracking subsystem handles IPv6 fragments without disturbing Path MTU discovery.

As a result of the solutions, connection tracking subsystem can track the state of a network flow even if a path of the flow is changed and turned back, and it can protect nodes from the packets with spoofed addresses in IPv6 extension headers under firewall running stateful filter.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firewall)*; D.4.4 [**Operating Systems**]: Communications Management

General Terms

Design, Security

Keywords

IPv6, connection tracking, Linux, stateful filtering, firewall

1. INTRODUCTION

Today stateful filtering is widely deployed in IPv4 networks. A firewall that runs stateful filtering to protect a local network can block unsolicited packets from outside the firewall. The firewall figures out whether incoming packets are related to any flow initiated by hosts in local network.

The mechanism of stateful filtering can be applied to IPv6 network. As recently many products and operating systems support IPv6 stateful filtering, Linux also support it. We implemented connection tracking subsystem on Linux that can track states of both of IPv4 and IPv6 network flows[7]. It is utilized by packet filter to perform stateful filtering.

To handle IPv6 packets, connection tracking subsystem has to take IPv6 specific semantics into account such as fragmentation, Type 0 Routing Header, and Mobile IPv6, because stateful filtering should not prevent the deployment of new IPv6 features. We solved the issue of handling fragmented packets in previous work, but more improvements are necessary for others. For example, a path of communication can be changed by using Routing Header or Mobile IPv6. The firewall running stateful filtering should not block packets on the path even if the path inspected by the firewall is changed and turned back. Moreover, stateful filtering should not open up any security issue by handling new features of IPv6.

In this paper, we propose solutions to issues that arise where connection tracking subsystem handles Routing Header and Mobile IPv6. We also describe a more concrete issue concerning Fragmented Header handling for connection tracking. It is not known well in previous work.

In the next section, we describe the design and implementation of connection tracking subsystem on Linux. In section 3, we describe issues in IPv6 packet handling and propose our solutions for them. In section 4, we present related work on stateful filtering, and we conclude in section 5.

IPv6'07, August 31, 2007, Kyoto, Japan.

2. CONNECTION TRACKING SUBSYSTEM

In this section, we describe the design and implementation of connection tracking subsystem, named **nf_conntrack**. We also describe how **nf_conntrack** handles IPv6 fragments. It was the biggest IPv6 specific issue when we implemented **nf_conntrack**.

2.1 Definition of Connection

nf_conntrack defines a connection that can be distinguished by the following parameters.

- The network address family
- The pair of network addresses on endpoints
- The transport protocol number
- The pair of identifiers defined by transport protocol, such as TCP port numbers

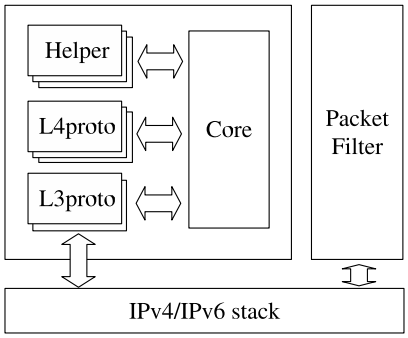

Figure 1: Module structure

A connection has 2 identifiers for both directions in the connection.

Some ICMP packets that represent request or reply also construct connections. For example, Echo Request and Echo Reply with the same network addresses and the same identifier of message belong to the same connection.

On the other hand, ICMP errors do not belong to any connection, but they are handled as packets related to the original connection which has caused errors.

Connections with unsupported transport protocol are distinguished by network addresses and transport protocol number.

2.2 States of Connection and State Table

nf_conntrack analyzes incoming packets to track states of connections. And it labels the structure storing the packets with one of the following. Each of them represents a snapshot of state at the time of analysis.

- **NEW**: New connection has been created by the packet.
- **ESTABLISHED**: Packets in both directions of the connection have been detected.
- **RELATED**: New connection has been created by the packet. This packet is expected as a result of analysis of another connection. For example of FTP, **nf_conntrack** expects new TCP connection for FTP data session by analyzing FTP commands on FTP control session.
- **INVALID**: Unexpected content have been detected in the packet.

nf_conntrack also tracks states of transport protocols. For example, it tracks TCP state and change of window size[14]. It can also count the number of packets in each connection.

All connection information is stored in the state table. It is implemented as a hash table whose size can be changed at any time. The default size is automatically adjusted depending on the total memory size.

2.3 Module Structure and Processing

Figure 1 describes module structure of **nf_conntrack**. It consists of modules categorized into 4 types.

- L3proto modules
- L4proto modules
- Helper modules
- Core module

An l3proto module handles network protocol specific part. It intercepts packet processing in network stack, reassembles fragmented packets, and parses the network protocol header. An l4proto module analyzes transport protocol header and tracks states related to the transport protocol. **nf_conntrack** has modules for TCP, UDP, ICMP, ICMPv6, SCTP, and GRE for PPTP. On the other hand, a helper module analyzes application data such as FTP, H.323, PPTP, IRC, NetBIOS, SIP, and TFTP. It also has a function that expects a new connection related to the tracked connection. The core module is the main part of connection tracking, and provides functions to manage states of connections to other modules.

nf_conntrack processes packets and connections as follows. At first, an l3proto module intercepts IP processing before routing and packet filtering. And it reassembles packets if they are fragmented. It also gets the offset to transport protocol header in it.

Next, the core module retrieves an identifier from the packet with the help of the appropriate l4proto module and the l3proto module. And it searches for an entry of the connection on the state table by the identifier. If the entry does not exist, the core module creates a new one.

The core module then labels the packet as described in section 2.2. This allows Linux kernel to process **nf_conntrack** and other subsystem such as packet filter in parallel. This is because packet filter can refer to the label for stateful filtering without referring to the state table. Then **nf_conntrack** can update states of the connection without disturbing packet filter.

The l4proto module and helper module analyze the packet and update states of the connection. If the helper module figures out that a new connection related to the connection will be created, it keeps the identifier of the new connection. If the expected packets have been detected, it is labeled with RELATED.

Each connection has an expiration timer that is refreshed when a packet belonging to it has been detected. When it expires, the connection is assumed to be disconnected and all information for it is destroyed. The expiration duration is decided by the l4proto module. For example, the TCP l4proto module sets it depending on the TCP state.

2.4 Usage Examples of Connection Tracking Subsystem

By using **nf_conntrack** and packet filter, users can filter packets according to the states of a connection. In Figure 2, an edge router has connection tracking subsystem and packet filter. The packet filter can figure out whether incoming packets from the Internet are solicited or not. The packet filter can achieve this by only checking the label of the packets. If ESTABLISHED is labeled to it, it is a reply packet to an outgoing packet sent from the inner network in the past. If RELATED is labeled to it, it is related to another connection. Then the router can pass only solicited incoming packets from the Internet, which are labeled with ESTABLISHED or RELATED.

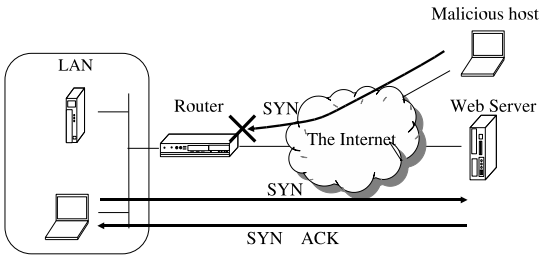

Figure 2: A usage example of connection tracking subsystem

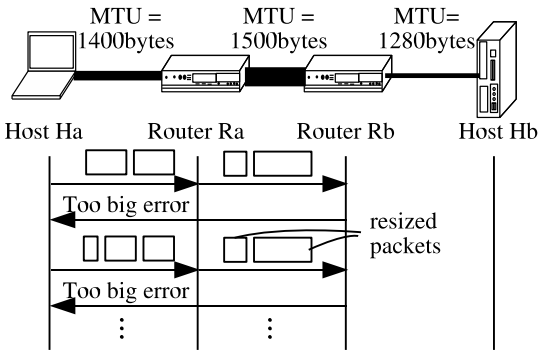

Figure 3: A case disturbing Path MTU Discovery

2.5 Handling IPv6 Fragments

In this section, we describe how **nf_conntrack** handles IPv6 fragments.

Helper modules of **nf_conntrack** analyze application data in packets. They might need to parse overall segment or datagram. If packets are fragmented, **nf_conntrack** needs to reassemble them. One possible solution would be to reassemble them before analysis, and fragment the reassembled packets after the analysis as we do in the IPv4 l3proto module. However, this solution would result in generating packets fragmented differently from the original ones.

This behavior is unacceptable for IPv6. The first reason is that it makes Path MTU Discovery unworkable in environments such as Figure 3. In this figure, only router Ra runs **nf_conntrack**. MTU of the link between Ra and Rb is larger than MTU of the link between Rb and Hb. If Ha sends fragments to Hb, the packets forwarded to Rb may be larger than MTU of the link between Rb and Hb. In that case, Rb returns ICMPv6 'Too Big' to Ha. However even if Ha resends them with smaller size, Ra keeps forwarding them with larger size and Rb can never forward them.

This problem can also arise in the case of IPv4 fragments with the Don't Fragment bit set. However it is more serious for IPv6 because a router never fragments IPv6 packets.

The second reason is that IPv6 stack on Linux does not expect to get reassembled packets from **nf_conntrack**. If we adopted this solution, it would required an undesirable number of changes in the whole IPv6 stack. In addition, the IPv6 specifications do not assume such behavior of a router[2], thus serious problems might raise due to future extensions to IPv6.

Our solution is as follows. If a packet is fragmented, the IPv6 l3proto module queues a light-weight copy of it. When the whole fragments are gathered, the IPv6 l3proto module reassembles the copied fragments. After analysis of the reassembled packet, the IPv6 l3proto module passes the original fragments to IPv6 stack. Because a router running **nf_conntrack** can forward fragments without modification, it can avoid to disturb Path MTU Discovery.

The reassembled packet is kept until all of the original packets are sent or discarded. This makes it possible for other systems to use it. For example, the packet filter can refer to the reassembled packet when the packet filter evaluates a rule that matches packets including a specific UDP port number. If a UDP packet is fragmented and the first fragment of it contains UDP header, the packet filter can utilize the reassembled packet to figure out the port number bound to the not-first fragments.

3. IMPROVEMENTS TO CONNECTION TRACKING SUBSYSTEM

In this section, we propose improvements to connection tracking subsystem so that it can handle IPv6 extensions.

3.1 Type 0 Routing Header

Type 0 Routing Header (RT0) is a IPv6 extension header, which contains intermediate addresses that the packet should visit before it reaches its final destination, and the final destination of the packet is contained in RT0. Even if a packet includes RT0, the current **nf_conntrack** uses destination address in IPv6 Header to identify connection. In addition to changing this behavior so that it can parse IPv6 Routing Header and use the final destination address, more consideration is necessary.

A connection path can be changed by using RT0. In such cases, **nf_conntrack** might not be able to handle all packets of connection on some routers running **nf_conntrack**. In this section, we present four cases and show what we can do to improve the behavior of **nf_conntrack** in each case.

The first three cases are that router Ra, Rb, or Rc is running **nf_conntrack** in Figure 4. The fourth case is that router Rb is running **nf_conntrack** in Figure 5. In Figure 4, host Ha and host Hb send TCP packets including a address of router Rb in RT0 for a while. After that, they send packets via router Rc without RT0. Ra is the default router for host Ha and all packets from/to host Ha pass through Ra. On the other hand, Figure 5 shows a situation where only the path on the way from host Hb to host Ha is changed.

The first case is that router Ra is running **nf_conntrack**. It can track states of connection during the whole period. There is no issue for **nf_conntrack** except for parsing RT0.

The second case is that router Rb is running **nf_conntrack**. It can track states of connection until its path is changed. If the hosts Ha and Hb turn back the path so that packets pass through router Rb, **nf_conntrack** handles them as invalid. This is because their sequence numbers are inconsistent with information tracked by **nf_conntrack**.

For a such case, we can use a configuration of the TCP l4proto module that loosens restriction on inconsistent sequence number. However, it does not restart window tracking but just ignores unexpected sequence number.

To track sequence number more strictly, we propose to add a function of the TCP l4proto module to restart tracking

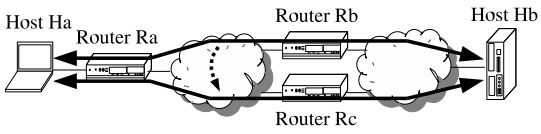

Figure 4: Changing path with Type 0 Routing Header

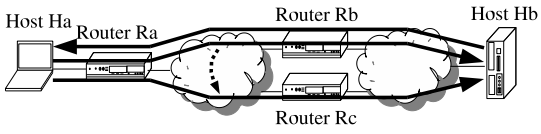

Figure 5: Changing path for one way with Type 0 Routing Header

sequence number. And we also propose to add a new module to packet filter so that packet filter can execute the function when a packet matches a filter rule configured by a user. By using the module and the module to detect RT0, packet filter can restart tracking sequence number only when it detects the packet containing RT0. User can also restrict the address in RT0 to prevent TCP l4proto module from restarting tracking sequence number by a packet containing spoofed address in RT0.

The third case is that router Rc is running **nf_conntrack**. When path is changed, **nf_conntrack** on Rc detects a packet of a new connection. It is the packet in the middle of TCP session, but the TCP l4proto module can start to track states. This mechanism is originally developed to cope with rebooted router. If the TCP l4proto detects a packet that is in the middle of TCP session, and if the packet does not belong to any connection in state table, the l4proto module can regard it as established.

One restriction in this case is that Rc drops packets from the host Hb until it detects any packet from the host Ha, where Rc is running stateful filtering to protect local network including the host Ha. Then whether this mechanism can be used is dependent on application.

The fourth case is that router Rb is running **nf_conntrack** in the Figure 5, and only the path on the way from the host Ha to the host Hb is changed. In this case, **nf_conntrack** cannot detect packets on the way from the host Ha to the host Hb after the path is changed. There is no way for Rb itself to keep tracking states of the connection.

3.2 Mobile IPv6

It is difficult or impossible for **nf_conntrack** to track states of connections in some cases where Mobile IPv6 (MIPv6) is used.

At first, MIPv6 has a mechanism called route optimization[6]. It makes possible for Mobile Node (MN) to directly communicate with Correspondent Node (CN) without passing data traffic through Home Agent (HA). This means that a path of a connection can be changed and there are similar problems described in the previous section.

Secondly, **nf_conntrack** cannot analyze encrypted packets. MN uses Encapsulating Security Payload (ESP) to protect signaling to HA, and it may also use ESP for data

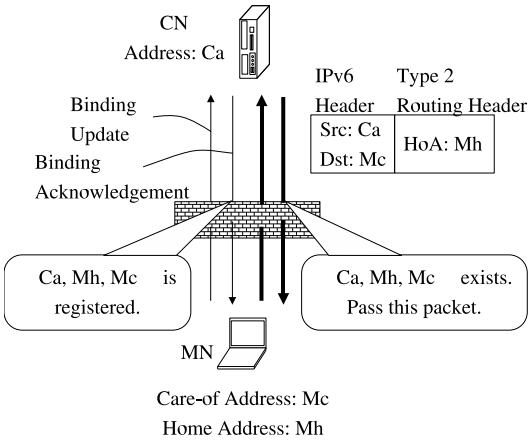

Figure 6: Tracking Binding Update

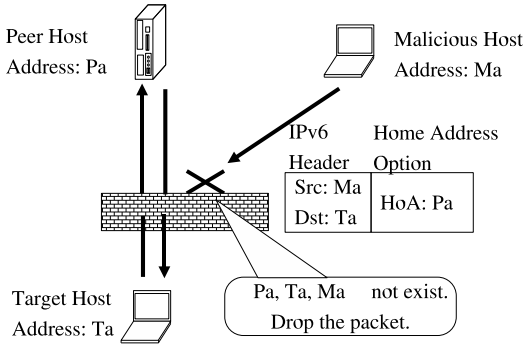

Figure 7: Blocking packets including spoofed address in Home Address Option

traffic on the path to HA. In that case, a router running **nf_conntrack** in the middle of the path cannot analyze contents in encrypted packets.Other problems with stateful filtering have been found[8].

Even though the above difficulties exist, we think that **nf_conntrack** is useful in some cases. The first case is that MN runs stateful filtering with **nf_conntrack** to prevent unsolicited access from other hosts.

To do this, the current **nf_conntrack** needs improvements so that it can handle Home Address Option (HAO) in Destination Options Header and Type 2 Routing Header (RT2). The former includes home address of MN. It is used when MN directly sends packets to CN. **nf_conntrack** needs to recognize it as the source address of the packet. The latter also includes home address of MN but it is used when CN directly sends packets to MN. The **nf_conntrack** needs to recognize it as the destination address of the packet.

The second case is that a firewall at the border of network enables stateful filter. By the improvement for the first case, the addresses in HAO and RT2 are used to identify a connection. A malicious node may send a packet including spoofed address in HAO or RT2 from outside the firewall. If an identifier retrieved from the packet matches an entry in the state table, the packet can pass through the firewall.

Figure 6 shows our proposal. **nf_conntrack** keeps the

valid tuple of home address of MN, care-of address of MN, and the address of CN from Binding Update (BU) and Binding Acknowledgement (BA). BU is sent from MN under the firewall to CN, and BA is replying packet to it. Then they can pass through the firewall. The care-of address of MN is stored in the entry of the connection. The other addresses are included in the identifier of the connection.

The packets from valid CN can pass through the firewall as shown Figure 6. When the IPv6 l3proto module in the firewall detects a packet with RT2 from CN, it searches an entry of the connection for BU whose endpoint addresses are the same as those of the packet. In the case of this figure, it can find the entry. Because the same tuple of addresses is stored in the entry, **nf_conntrack** marks the packet as valid. As a result, packet filter allows it to pass through.

On the other hand, Figure 7 shows the case that a malicious host tries to pass a packet with spoofed HAO through a firewall. An entry that matches the identifier derived from the packet exists in the state table. However there is no entry for BU related to the packet. Then **nf_conntrack** can figure out that it is an invalid packet. As a result, packet filter drops it.

Two changes are required to realize this mechanism.

- A new l4proto module for Mobility Header to analyze BU and BA.
- Changes to the IPv6 l3proto module to search for an entry for BU in state table.

There are some restrictions to this solution.

- If all packets for BU between MN and CN are encrypted, **nf_conntrack** cannot analyze them[3].
- The cost of searching for an entry of a connection is expected to be large. More efficient structure of state tables is required.
- If BU from MN is fake, unsolicited incoming packets might pass through the firewall.

The last restriction is because this mechanism trusts behavior of MN under the firewall. However such trust model is common with stateful filtering for other protocols. Another mechanism is required under the trust model that does not trust nodes under the firewall.

4. RELATED WORK

OpenBSD has a stateful packet filter which is named **pf**[5]. **pf** can also make decision on state of connection. Unlike **nf_conntrack**, state table is implemented as AVL tree that is a balanced binary search tree. It can reassemble IPv4 fragments. Like **nf_conntrack**, **pf** does not keep the size of the original IPv4 fragments and they might be resized to the MTU size of output interface. On the other hand, **pf** does not support IPv6 reassembly and IPv6 fragments do not affect any state. **pf** always uses source address and destination address in IPv6 header for the identifier like the current **nf_conntrack**. Routing Header and Destination Options Header containing Home Address Option do not affect identifier of connection.

IPFW2 of FreeBSD and **IP Filter** also support stateful filtering[1]. However they do not reassemble IPv6 fragments and do not specially handle Routing Header and Destination Options Header, as far as we studied. Instead, **IP Filter** specially handles IPv4 fragments in the other way. It caches the result of packet filtering for the first detected fragment. And it applies the result to other fragments that have the same identifier in IPv4 header as the cached one. This makes it efficient to handle fragments because it is unnecessary to gather and reassemble all fragments. However users need to aware the fact that the result of packet filtering might be different depending on the order of processing of the fragments. On the other hand, packet filter on Linux and **pf** applies filter rules to reassembled packets. Then the result of packet filtering is not affected by whether fragments are reordered or not.

There is an another proposal to IPv6 stateful filtering. Orla McGann and David Malone proposed using the randomly generated value in Flow Label field of IPv6 Header as an another piece of state of a TCP connection[10]. It keeps the value of Flow Label in SYN packet and SYN-ACK packet, and drops packets in the TCP connection if the value has been changed. They also proposed an option that allows the value being changed only once for some operating systems which changes the value after the 3-way handshake. Their feature can prevent TCP packets from passing through firewall even if their TCP headers are spoofed. **nf_conntrack** does not track the value of Flow Label field because it may vary even in a single TCP connection. However it is not difficult to add support for Flow Label tracking to **nf_conntrack**, because IPv6 l3proto module is enough to keep the value of Flow Label in IPv6 packets in a connection.

This paper describes restrictions of **nf_conntrack** and stateful filtering. Most of them are because there is no way for **nf_conntrack** to get enough information about future communications. There are some approaches so that a node can directly provide information about future communications to firewall. The Next Step in Signaling (NSIS) working group of The Internet Engineering Task Force (IETF) is standardizing signaling protocols for QoS control. And a protocol for NAT and firewall traversal is also being developed [13]. Universal Plug and Play Internet Gateway Device (UPnP IGD) also has a feature to control Internet Gateway Device so that specified incoming packets can pass through it[4]. Annex describes how UPnP messages over IPv6 are handled, but there is no specification to control IPv6 filter rules on Internet Gateway Device.

Some approaches was proposed to solve incompatibility between firewall and Mobile IPv6. Yang Shen et al. proposed a firewall between MN and CN to permit the packets containing the Mobility Header, and to parse RT2 and HAO[12]. They also proposed a mechanism to detect firewall between MN and HA to figure out whether UDP encapsulation is necessary or not. However they provides no solution to block packets containing spoofed addresses in HAO and RT2. Our solution provides stricter policy and can block such spoofed packets from outside the firewall as described in section 3.2

Pan Jian Li and Chen Shan Zhi proposed an extension to Diameter so that AAA server for Mobile IPv6 service can control firewalls where the AAA server and the firewalls are operated by the same service provider[9]. This solution makes MN possible to completely traverse firewall without any extension to MN, and suitable in such network model. However it does not work in the network model where a private firewall exists, which is uncontrollable from the AAA

server. On the other hand, our solution can be applied to a private firewall with some restrictions as we described in section 3.2.

5. CONCLUSION

In this paper, we proposed improvements for connection tracking subsystem on Linux so that it could handle Routing Header and Home Address Option. The described issues and solutions described can be categorized as follows.

- Unsupported extension headers. The solution was simply to use addresses in Routing Header and Home Address Option to identify connection instead of addresses in IPv6 header.
- Changing a path of connection. **nf_conntrack** originally had features to solve some of the problems. In addition to it, we proposed a way to restart tracking in the case that the path was changed and was turned back. The remaining issue is that the incoming packets cannot pass through a firewall running stateful filter until outgoing packets pass through the firewall after the path is changed.
- New problems caused by supporting IPv6 extension headers. We proposed a solution to protect a node under the firewall running stateful filtering from the incoming packets with spoofed address in Type 2 Routing Header or Home Address Option.
- Encrypted packets for the extensions. There is no way to analyze their contents in them if **nf_conntrack** is running in the middle of communication path.

We also described how **nf_conntrack** handles IPv6 fragments without disturbing Path MTU Discovery.

nf_conntrack is available on Linux kernel 2.6.14 or later, and mainly maintained by Netfilter Project[11]. As a result of contribution by many developers, all of the features related to **nf_conntrack** such as IPv4 NAT, the application programming interface to control **nf_conntrack**, and many helper modules have been re-implemented. So, the original one will be removed from Linux kernel 2.6.22.

We introduced IPv6 fragments handling when we implemented **nf_conntrack** in the previous work. Other ideas in this paper are still not implemented on Linux kernel and they are future work. However we believe that it is possible to improve IPv6 packet handling of **nf_conntrack** with our solutions. Moreover, we should evaluate the impacts of our proposals to **nf_conntrack**. Especially, our mechanism to block spoofed Mobile IPv6 packets requires to search the entry in state table and its impact might be large.

Acknowledgement

This work is supported by the WIDE/USAGI Project. I would like to thank Harald Welte, Patrick McHardy, and other developers related to Netfilter Project for spending their time to review my contribution.

6. REFERENCES

[1] IP Filter Homepage. http://coombs.anu.edu.au/~avalon/.

[2] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC2460, Devember 1998.

[3] F. Dupont and J-M. Combes. Using IPsec between Mobile and Correspondent IPv6 Nodes. Internet draft, February 2007.

[4] UPnP Forum. Internet gateway device (IGD) standardized device control protocol V 1.0, November 2001. http://www.upnp.org/.

[5] D. Hartmeier. Design and Performance of the OpenBSD Stateful Packet Filter (pf). In *Proceedings of the the FREENIX Track: 2002 USENIX Annual Technical Conference*, June 2002.

[6] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC3775, June 2004.

[7] Y. Kozakai, H. Yoshifuji, H. Esaki, and J. Murai. Design and implementation of IP version independent Connection Tracking System on Linux. In *Technical report of IEICE*, number IN2004-248, pages 29–33, February 2005. (in Japanese).

[8] F. Le, S. Faccin, B. Patil, and H. Tschofenig. Mobile IPv6 and Firewalls: Problem Statement. RFC4487, May 2006.

[9] Pan Jian Li and Chen Shan Zhi. A Mobile IPv6 Firewall Traversal Scheme Integrating with AAA. In *WiCOM 2006.*, September 2006.

[10] O. McGann and D. Malone. Flow Label Filtering Feasibility. In *European Conference on Computer Network Defense 2005*, December 2005.

[11] Netfilter Project. Netfilter Project Web Page. http://www. netfilter.org.

[12] Yang Shen, Zhang Sidong, Zhang Sidong, and Miao Fuyou. Firewall Traversal for Mobile IPv6. Internet draft, November 2005.

[13] M. Stiemerling, H. Tschofenig, C. Aoun, and E. Davies. NAT/Firewall NSIS Signaling Layer Protocol (NSLP). Internet draft, March 2007.

[14] G. v. Rooij. Real Stateful TCP Packet Filtering in IP Filter. In *10th USENIX Security Symposium*, August 2001.